Axiom-Pinpointing in Description Logics and Beyond

Dissertation

zur Erlangung des akademischen Grades
Doktor rerum naturalium (Dr. rer. nat.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
M. Sc. Rafael Peñaloza Nyssen
geboren am 5. Juli 1981 in Mexiko-Stadt

verteidigt am 14. August 2009

Gutachter:
Prof. Dr.-Ing. Franz Baader,
Technische Universität Dresden
Prof. Ulrike Sattler,
University of Manchester

Dresden, im Oktober 2009

Contents

To those who taught me
the value of contradictions.

Acknowledgements

This work was financially supported by the *Deutsche Forschungsgemeinschaft* (DFG) under grant GRK 446. In this respect, the author wishes to thank Prof. Dr. Gerhard Brewka for the oportunity of working at the *Graduiertenkolleg Wissensrepräsentation*.

Chapter 1

Introduction

Explanations are an essential component for the development of science. Very roughly, scientific progress can be divided into two steps, each having a close connection to a different interpretation of the word *explanation*: finding a theory that explains a set of observations, and explaining why a given fact can be deduced from a specific theory.

When confronted with a set of observations, one can try to produce a general theory that *explains* them; in other words, one from which all such observations are a consequence. The adjective *general* anteposed to the word *theory* is intended to express that this theory can be used to deduce not only the given observations, but also additional, possibly previously unknown, facts. These additional facts allow for our theory to be tested, by designing experiments that confirm or contradict them. The theory becomes stronger with each new observation that confirms it, but the moment one contradicting observation is found, the theory needs to be refuted and replaced by a new one that accounts also for this observation.

A refuted theory needs not be totally wrong; indeed, it is possible that only a minor portion of the whole theory is responsible for the contradiction between the deduced facts and the new observations. Instead of creating a new theory from scratch, one can try to remove the wrong portions; that is, those from which the contradicted facts can be deduced, and then extend this theory to account for all the observations that do not follow anymore from the reduced theory. Finding the wrong portions of the theory can be seen as *explaining* the contradicted facts, within the context of the theory.

One famous example of this process is the discovery by Johannes Kepler of the elliptical shape of planetary orbits, as described in his *Astronomia Nova*. Using the very precise and methodic measurements of the position of the planet Mars made by Tycho Brahe during his lifetime, Kepler found a displacement of up to eight minutes of a degree with respect to the position predicted by the astronomic theory of the time. Convinced of the precision of the measurements, this admittedly small displacement prompted him to correct the theory. His first step consisted on showing that a circular orbit was incompatible with Brahe's observations, thus distinguishing the hypothesis of circular planetary motion as the source of the disparity between the theory and said observations. Keeping the rest of the astronomical theory intact (for instance, still assuming that the sun was a stationary body in space around which all planets

traveled) Kepler needed only to find a shape for planetary orbits that agreed with the set of observations he had. After trying with different ovoid shapes, he finally settled that an ellipse with very low eccentricity and the sun standing at one of its foci, best described the path followed by the planets. This discovery is nowadays known as his First Law of planetary motion.

It is perhaps worth noticing at this point that the term *observation* is being used in a very loose manner that can express factual observations, such as the position of Mars at a given moment in time, as well as more general theories. For instance, Isaac Newton's law of universal gravitation can be seen as a general theory explaining, among other observations, Galileo's law for free fall of bodies and Kepler's first two laws of planetary motion.

The importance of explanations in science has been long known: it can be tracked back at least as far as Aristotle's *Posterior Analytics*, with more recent examples including Karl Popper [Pop35] and John Stuart Mill [Mil43]. But it was only after Hempel and Oppenheim's logic-based theory of explanation [HO48] that the topic received a wider interest and was treated in a formal and methodical manner. The work by Hempel and Oppenheim focuses on the first kind of explanations described above, which is called *scientific explanation* in modern Philosophy of Science: given an observation E, a theory T *explains* E if E can be derived from T and there are no superfluous elements in T; in other words, if there is no subtheory T' of T from which E can also be derived. In this case, E is called the *explanandum* and T the *explanans*.[1] What distinguishes [HO48] from previous studies on scientific explanations is the agnisation of the need for a formal definition of the terms *theory, observation*, and *derivation*. To this end, the authors propose a language based on first-order logic, in which the *explanandum* and *explanans* need to be represented, yielding logic-based formal semantics to the ideas of scientific explanations: theories and observations are sets of formulae and formulae in this language, respectively, while derivation is given by the standard notion of logical entailment.

Soon, this theory of scientific explanations began to be strongly criticised due to its excessive generality. It is interesting that most of these criticisms were not aimed to the intuitive notion of scientific explanation, but rather to the representational language used in their formalisation. Paradigmatic examples of this view are the trivialisation theorems [EKM61]. Roughly, these theorems show that given almost any arbitrary sentence E and theory T, it is possible to construct a theory T', derivable from T that works as an *explanans* for E. In words, what these results say is that when confronted with an observation, one can first construct any arbitrary theory, totally unrelated with the given observation, and from it build an explanation satisfying Hempel and Oppenheim's notion. Several efforts have been done to solve this problem by either restricting the representation language, or imposing additional conditions in the set of formulae that form an *explanans*. Hempel himself spent twenty years fine-tuning both, his representation language, and the notion of what is an *acceptable* explanation [Hem65].

In reality, the trivialisation theorems are less surprising than it might look at first

[1] For a survey on the origins and first developments of scientific explanation, see [Sal89, Sch96].

sight. The language introduced in [HO48] is intended to solve two problems simultaneously: knowledge discovery, and knowledge representation. As a consequence, the representation language needs to be able to describe any conceivable *explanans* for any conceivable *explanandum*. We aim at a fairly less ambitious goal, where the knowledge discovery problem has been solved already; we will nonetheless rely on the same notions of explanation, in dependency with the knowledge representation formalism chosen.

Knowledge representation deals with the problem of storing the knowledge of a domain in an efficient and usable manner. The search for a solution to this problem obtained special attention from the second half of the past century as an important milestone for the area of Artificial Intelligence. In a nutshell, before a machine is able to show any intelligent behaviour, it needs to have a mechanism for storing and manipulating pieces of knowledge. The stored knowledge is usually called a *knowledge base* or *ontology*. Rather than having a knowledge base explicitly stating every piece of knowledge, one would prefer to be able to infer additional information that appears implicitly in this knowledge base. For instance, knowing that Albert is a Human, and that all Humans are Mammals, it should not be necessary to additionally express that Albert is a Mammal, as this is a direct consequence of the other two pieces of knowledge. Our representation language should thus be accompanied by an inference engine that allows the user to make such facts explicit.

Two early knowledge representation formalisms are Semantic Networks [Qui67], developed by Quillian, and Frames [Min81] introduced by Minsky. The main drawback of these formalisms is their lack of a formal semantics by which the knowledge represented in them can be unambiguously interpreted. Hence, it was impossible to construct a system that could infer knowledge from arbitrary knowledge bases. A system developed for working on such ontologies required to make choices regarding the semantics of some of the constructors, which made it usable only in the specific application it was developed for. Description Logics arised as a way to alleviate this problem, using some of the main ideas of Semantic Networks and Frames, but giving them formal and easy to understand semantics.

Description Logics [BCM$^+$03] are a family of logic-based knowledge representation formalisms with clear and well-defined semantics, built in most cases as sublanguages of first-order logic. The family covers a wide range of expressivity levels, with their corresponding trade-off in complexity. On the lower part of the expressivity scale is the description logic $\mathcal{EL}$ whose relevant inference problems are decidable in polynomial time [Baa03c, Bra04b]. This logic has been successfully applied to represent knowledge from the biological and medical fields [Sun09]. A fairly more expressive description logic is $\mathcal{SHOIN}(\mathcal{D})$, the one behind the Web Ontology Language OWL, which was selected by the World Wide Web Consortium as the representation language for the Semantic Web [HPSvH03]. Although the inference problems for this logic turn out to be intractable, highly optimized reasoners have been shown to behave well in practice [HST00, HS04].

The existence of a formal (and recommended) language motivated people to start constructing realistic ontologies and reasoning with them. Successful stories rapidly

triggered the proliferation of more and larger knowledge representation efforts. As the size of these knowledge bases rapidly increases, the need of automatic explanation and correction tools becomes much more obvious. Indeed, ontology development is, just as software development, an error-prone activity. Since large ontologies are typically developed by groups of experts, clashes in their individual views may account for the existence of errors. On the one hand, it is not uncommon to find experts disagreeing in particular aspects of the area being represented. Such disagreements can easily provoke the insertion of contradictory information to the knowledge base. On the other hand, even if all experts concur on the knowledge being modeled, they can still dissent on the way it should be translated to the representation language. This is deeply related to the problem of expertise: usually, experts in the domain field are not experts in knowledge representation, and *vice-versa*. An ideal ontology development group should be proficient in both areas. Furthermore, with large ontologies it is usually hard to predict the whole effect of a minor variation, which can easily lead to unexpected, if not necessarily erroneous, consequences. Finally, representation choices are sometimes made but not used uniformly or adequatedly along the whole ontology.

In any of these cases, it is desirable to track back to the specific portion of the knowledge base that is responsible for a given consequence. In other words, we are interested in finding *justifications*: given a consequence E of an ontology T, a portion T' of T *justifies* E if E is a consequence of T' and E is not a consequence of any strict portion of T'. Obviously, for this definition to make any sense, one needs to be able to divide the full ontology in smaller parts. We will give the name *axiom* to the indivisible segments of the knowledge base. Notice that the definition of justification corresponds exactly to the second notion of explanation presented at the beginning of this chapter.

Although finding justifications by hand may be feasible for very small ontologies, performing this task without the help of an automated tool seems unrealistic once the border of the hundreds of axioms has been crossed; much more for ontologies of the kind of SNOMED CT [Spa05, SPSW01] which has over 340 000 axioms. The current version of SNOMED CT classifies the *amputation of a finger* as a subconcept of *amputation of hand*. In other words, according to this ontology, someone who has an amputated finger has also suffered the amputation of a hand.This erroneous inference follows from only six axioms of the ontology, and is caused by an erroneous use of a representation schema developed for describing the transitivity of some properties [BS08].[2]

A justification distinguishes precisely those elements of an ontology that are responsible for the derivation of a given consequence E. If E is known to be erroneous, then justifying it means to detect the sources of this error; with this knowledge we can then correct the ontology and get rid of E. But one should not forget that a single consequence may have more than one justification in the given ontology. In order to ensure that E is not a consequence of the corrected ontology, one would have to account for each of these justifications. Alternatively, we can try to find a *diagnose* for

[2] In fact, the same problem with transitivity presents itself in more than one example in SNOMED CT; for instance, *amputation of hand* is also classified as a subconcept of *amputation of arm*.

E: a minimal portion of the ontology T such that, if removed from T, E is no more a consequence. Returning to our original example, Kepler diagnosed that the source of the disparity between the theoretically-predicted and the experimentally-found positions of Mars was the assumption that planets follow a circular orbit. Removing this assumption from the astronomical theory led to a theory without the unwanted disparity. This theory, nonetheless, also was unable to predict the position of any planet at any time, nor even eclipses or other important astronomical events. In the process of removing an unwanted consequence, we can easily get rid also of *wanted* consequences; hence the need for a diagnose to be minimal, ensuring this way that the pruning of the ontology produces as small a change as possible.

Recalling the notion of scientific explanation, one can easily confirm that a justification for a consequence E is in fact a scientific explanation for E (seen as an *explanandum*) where the sentences of the *explanans* are restricted to belong to the original ontology. Conversely, it is possible to see the construction of an ontology as the result of knowledge discovery, in which case a scientific explanation for E is in fact a justification for E over the newly generated ontology.[3] Notice that neither notion of explanation really depends on the representation language used. This in particular shows that, although much effort has been set in discrediting and fixing Hempel and Oppenheim's notion of scientific explanation, along with the logic-based representation language they use, it is not the language, nor the theory *per se* that cause the main problems of this approach, but rather the intermediate task of knowledge discovery, where any arbitrary set of sentences can be used as an *explanans*. Any language with sufficient expressivity would be trivialisable under such a general attempt for explanations.

With the advent of newer representation languages, the original language described at [HO48], as well as its improved versions, remains relevant not so much as a knowledge representation formalism, but as a paradigm for the properties that a language must satisfy before a notion of explanation can be well defined over it. First, this language must be able to express two kinds of sentences: axioms and consequences, having formal semantics. Additionally, a notion of derivability of a consequence from a set of axioms is necessary. Since the definition of explanation requires a *minimal* portion of the ontology from which the consequence follows, derivability must be monotonic in the sense that growing the knowledge base will only add more consequences without removing any of the previously existent; otherwise, minimality makes no sense at all. Since first order logic is monotonic, so is Hempel and Oppenheim's language, and thus is this condition implicitly satisfied; nonetheless, once we decide to work with a distinct language, this condition must be ensured. Finally, Hempel did realise that not every set of axioms can be considered a theory: it might be necessary to ensure an internal coherence between the axioms used. The notion of coherence may obviously

[3] A small, but important, distinction is in order. In scientific explanation one will usually consider a fixed background theory over which the new theory is being built. Justifications, on the other hand, usually consider each axiom as refutable, in order to obtain the real source for the deduction. This description of scientific explanation is closely related to the idea of *abduction* in Artificial Intelligence. In this case, knowledge discovery would try to find a set of plausible axioms, called *abducibles*; a theory is then extended with a minimal set of abducibles to entail the observations.

change between languages. Thus it is not only necessary to define axioms in a specific language but also which sets of them are admissible as ontologies.

A desirable property of any knowledge representation formalism is the ability of implicitly encoding some pieces of knowledge that can then be summoned by an automatic procedure. In general, we want to have a way to know whether a specific sentence is a consequence of an ontology; in other words, we require a procedure that decides derivability. The answer that such a procedure yields obviously depends on both, the possible consequence, and the ontology under consideration.

To accommodate a theory of explanations, we need to make some assumptions on the theories and the notion of derivability used. First, we assume that any theory can be divided in parts, each of which is itself a theory that can be used as an input for the decision; in other words, a theory is composed of subtheories. As said before, we give to indivisible theories the name of *axioms*. Second, we require derivability to be monotonous; that is, if E is derivable from a theory T, then it is also derivable from any supertheory of T. Minimal subtheories from which an *explanandum* E is derivable are its *explanans*.

In this work we aim to develop methods for automatically finding justifications and diagnoses for consequences of a theory. Instead of working directly on the representation language, we consider derivability via a given decision procedure that is correct for a monotone notion of derivability. Obviously, explanations depend on derivation, and thus indirectly also on the procedure used for deciding it. We will hence try to transform a given decision procedure into an *explanation procedure* whose outputs are not yes or no, but an encoding of all its justifications or diagnoses.

Decision procedures can take a wide variety of forms, and trying to encompass all of them in our theory of explanation would be a titanic task. Hence, we focus on two prominent approaches: tableau-based and automata-based decision procedures. These two approaches have been widely used in description logics, and other areas, where their distinct complexity and efficiency properties have been exploited. But, although we will also use description logics for motivating our ideas and definitions, the applicability of our framework is not limited to these particular knowledge representation formalisms. We will, for instance, also show its applicability to linear temporal logic (LTL).

In a nutshell, tableau-based decision procedures start with some explicit knowledge translated from the input, and extend it with the application of rules depending on the theory, deriving the facts that are implicitly encoded in the input. The decision depends on the explicit knowledge present once the execution of the algorithm halts, by testing for so-called *clashes* in the knowledge produced. Automata-based decision procedures, on the other side, translate the input into an automaton $\mathcal{A}$ from which a decision is made depending on whether the language accepted by $\mathcal{A}$ is empty or not. The emptiness test of automata tries to disprove that this language is empty, but without actually building any element that would belong to it.

It should be noted that in general tableau-based procedures can decide a wider range of problems than their automata-based counterparts. This follows from the latter being limited to accepting languages of objects having a specific shape, while

the rule-based expansion of the former allows for a wider range of options. On the other hand, the arbitrary shape of structures constructed makes it harder to ensure even that the procedure will ever terminate, and in some cases appropriate techniques are necessary to avoid infinite expansions. This generality will force us to look deeper into tableau-based decisions and explanations, and ultimately restrict them to make sure that an answer will be found in finite time.

Structure of the Work

This work is divided as follows. We first dedicate Chapter 2 to the introduction of description logics and the temporal logic LTL as well as their main decision problems, along with tableau-based and automata-based algorithms for solving them. These algorithms will be used in the next three chapters to motivate our approach to automated explanations. The chapter is meant as a practical introduction to tableau- and automata-based decision procedures and their associated techniques; as such, the chapter summarises relevant portions of [BS01, BHP08, WVS83].

Chapters 3 and 4 deal with the tableau-based approach. The former chapter formalises first the notion of a tableau-based decision algorithm, what we call a *general tableau*, that receive as input an ontology and a sentence, and decide whether the sentence is derivable from the ontology. Our notion of general tableaux covers also some algorithms that are historically not considered to be tableau-like such as resolution [Rob65, Lei97], congruence closure [NO07], and the subsumption algorithm for the Description Logic $\mathcal{EL}$ [BBL05]. We then show how to change these general tableaux to obtain an algorithm that computes an encoding of all explanations of the input sentence within the input ontology. Our encoding will be through a so-called *pinpointing formula*: a monotone Boolean formula whose minimal satisfying valuations have a one to one correspondence with justifications. Finally, we show that our approach has problems with termination, in the sense that the algorithm proposed may not be able to yield a pinpointing formula in finite time.

In the latter chapter we try to solve the problem of termination by taking from the ideas of terminating tableaux used in description logics, which mainly exploit the tree-shape of the generated models. Termination is achieved in two different ways. First, we introduce a sub-class of tableaux whose so-called pinpointing extension always terminates without the need of any special stopping mechanism. Afterwards, we focus in formalising a notion of blocking: a method that allows us to detect cyclic computations and accordingly stop the execution of the algorithm without harming its correctness. The introduction of blocking to the tableau framework forces us to adapt the pinpointing extention in an appropriate fashion. Thus, correctness needs to be proved again for this variant setting. The ideas and results of these two chapters were first published in [BP07, BP09].

We then change our attention in Chapter 5 to the automata-based approach. Given an automaton deciding a property, we show how to construct a *weighted* automaton whose so-called behaviour is a pinpointing formula. We then show a bottom-up method for computing this behaviour in time polynomial on the size of the automaton.

The results presented here were originally published in [Peñ08, BP08] for the special case of looping automata. Here we present an extended version that can deal with generalised Büchi automata and a wider range of restrictions.

Before giving our conclusions and brief ideas for future work, we finish in Chapter 6 with an analysis of the complexity of explanation divided in three parts: first we show the complexity of finding justifications; then, we show analogous results for finding diagnoses, and finish the section by showing that the pinpointing formula cannot, in general, be represented in space polynomial on the size of the input ontology. These complexity results extend those shown for justifications and claimed to hold also for diagnoses in [BPS07a, BPS07b]. We then return to the tableau-based approach to show that it is impossible to fully characterise the class of all tableaux having a terminating pinpointing extension.

Related Work

The study of justifications in Description Logics has only recently begun. To the best of our knowledge, the first attempt to compute the justifications for unwanted consequences of a DL ontology was done by Schlobach and Cornet. In [SC03], the authors show that the standard tableau algorithm for the DL $\mathcal{ALC}$ [SS91] that decides satisfiability w.r.t. so-called *unfoldable* terminologies, can be extended with labels that keep track of the axioms responsible for an assertion to be generated during the execution of the algorithm.[4] They also coin the term *axiom pinpointing*, which we continue to use, to describe this task. Later on, Schlobach [Sch05] showed that diagnoses can be computed from the set of all justifications by means of a Hitting Set computation, following Reiter's Theory of Diagnoses [Rei87].

The problem of finding justifications and diagnoses in a DL knowledge base was actually considered one decade earlier in a different context. In [BH95], Baader and Hollunder consider the problem of extending DLs with default rules, which they solve by introducing a labeled extension of the tableau-based consistency algorithm for $\mathcal{ALC}$ w.r.t. ABoxes [Hol96]. The two labeling approaches, namely [BH95] and [SC03], follow very similar ideas. Factoring for the different kinds of axioms considered, the main difference between the algorithms is the shape of the output: while the algorithm in [SC03] yields all the justifications directly, the one by Baader and Hollunder outputs a monotone Boolean formula, from which all the justifications can then be deduced.

The two approaches have since then been extended to allow for more expressive languages. On one hand, Schlobach and Cornet's method [SC03] was extended by Parsia *et al.* [PSK05] to DLs using a wider variety of constructors. On the other, the ideas of [BH95] were extended by Meyer *et al.* [LMP06] to the case of $\mathcal{ALC}$ terminologies that use general concept inclusion axioms, which are no longer unfoldable. In [HPS08] the idea is further extended to deal with *portions* of axioms, to allow for a more precise understanding of the causes of derivability. In reality, the use of the DL $\mathcal{ALC}$ in both of the original approaches [BH95, SC03] was intended to work as a prototype that could be easily extended to other DLs with a tableau-based decision

[4] In this case, the unwanted consequence was the unsatisfiability of a concept name.

procedure. However, the extension in [LMP06] showed that some techniques used in tableau algorithms, such as blocking, require special attention when building their labeling extension to keep correctness. Our tableau-based approach to pinpointing tries to show how the same ideas can be applied in a more general setting.

In our general approach we faced the problem of how to ensure that the pinpointing algorithm will terminate in finite time. This problem arises already for tableau-based decision procedures, and it is directly inherited by their pinpointing extensions. A general solution for decision algorithms was proposed in [ST08, ST07] in which a rule is added to the tableau and always eagerly applied. This solution is not satisfactory for us, as we want to allow any possible ordering for rule application in both, the original tableau and its pinpointing extension.

All the previously cited approaches belong to the category of *glass-box* methods, in which the decision algorithm needs to be tempered with to create the algorithm that outputs all justifications. Since modern DL reasoners [HM01, Hor98, SP04] use several optimizations that cannot be applied to the labeling extension, recent research has also looked at ways of computing justifications using (unmodified) reasoners as a subprocedure. Most of these so-called *black-box* methods [BS08, KPHS07, SHCH07, Sun09] use a variant of Reiter's Hitting Set algorithm [Rei87], while trying to minimize the search space by either syntactical or semantical conditions. The black-box approach has the clear advantage of being able to use the most efficient reasoner available without bigger implementation problems; however, this reasoner may need to be called an exponential number of times before all justifications are found. Trying to couple the advantages of both glass-box and black-box algorithms, a mixed approach has been considered for the $\mathcal{EL}$ family of DLs. This mixed approach uses a glass-box method to compute a small (possibly non-minimal) set of axioms from where the consequence still follows, which is later minimized using black-box techniques [Sun09].

Although automata-based decision procedures have been widely used in the DL community [BHP08, BT01, CDGL99, CDGL02, LS00],[5] there has been no prior attempt to construct a glass-box pinpointing algorithm based on the automata approach. For our automata-based pinpointing framework, we had to construct, and compute the so-called behaviour of, weighted automata on infinite trees. Surprisingly, study on the area of weighted automata on infinite trees has only very recently begun, with its origin at [DKR08, KL07]. As a result of this, we needed to develop our own algorithm for computing the behaviour of these automata. Since the beginning of our work with automata, a different algorithm was developed independently by Droste *et al.* [DKR08]. We will show that, when applied to pinpointing, the algorithm in [DKR08] is actually equivalent to a naïve black-box method.

The problem of axiom pinpointing has arisen, usually with different names, in several distinct research areas. The SAT community has considered the problem of computing maximally satisfiable and minimally unsatisfiable subsets of a set of propositional formulae. Solutions to this problem include black-box approaches that call a SAT solver [BS05, LS05], as well as glass-box methods that extend a resolution-

[5] Up to now, automata-based procedures are used mainly for proving theoretical results in DLs. However, reasoners based on an automata-based algorithm for the temporal logic LTL have been successfully used in practice for Model Checking [GO01, GPVW95, Hol97].

based SAT solver [DDB98, ZM03]. In Linear Programming, several people have been interested in finding irreducible infeasible sets (IIS): minimal subsets of linear restrictions that have no solution. Several methods exist that compute *one* IIS [Chi97, CD91, TMJ96] using a black-box method. To the best of our knowledge, there is no glass-box approach to solving this problem. A different idea was presented by Gleeson and Ryan [GR90], showing that there is a bijection between the set of IIS and the optimal solutions of a *dual* linear programming problem. This idea was later employed by Bruni [Bru05] to find all minimally unsatisfiable subsets from a set of propositional formulae.

Another area where computing justifications has a special interest is Satisfiability Modulo Theories (SMT) (see, for instance [ACGM04, BBC$^+$05, GHN$^+$04]). SMT tries to find satisfying valuations of propositional formulae where each propositional variable represents a restriction from a background theory. Modern SMT solvers use a glass-box approach to find a single (possibly non-minimal) conflicting set of restrictions that voids the current valuation in as short a time as possible [NOT06].

Additionally from DLs, we use the temporal logic LTL to exemplify our automata-based approach. We view the conjuncts of an LTL formula as axioms and the justifications are minimal unsatisfiable subformulae that allow us to understand the overall unsatisfiability of the original formula. Although this setting seems not to have been considered for LTL before, it is closely related to the problem of computing unsatisfiable cores that has appeared in the SAT community [LS04].

As it was readily mentioned, the task of finding justifications closely resembles that of abduction. Abduction uses a background theory and an additional set of axioms called abducibles. The reasoning task consists in finding minimal sets of abducibles that, when added to the background theory, entail a given query. Abduction has been studied in several fields, but of special importance for this work is its application to propositional logic (for instance, de Kleer's ATMS [dK86a, dK86b, dK86c]), and in particular the complexity results that have been obtained for Horn formulae [EG95a, EM02]. We will use a similar approach for several of our complexity results in Chapter 6. Recently, the problem of abduction has also been considered in the DL $\mathcal{EL}$ [Bie08].

It is important to notice that for really *understanding* a consequence, computing justifications and diagnoses is usually insufficient. Individual axioms may be already hard to interpret, and the relationship between them far from obvious. In the former case, one would like to highlight the specific portions of the axiom that play a role in the derivation of the consequence [HPS08]; in the latter, one can try to combine several axioms in a single, easier to understand, new axiom also called *lemma* [HPS09].

Chapter 2

Logics and Decision Procedures

The main goal of this chapter is to describe, by means of examples, two of the most prominent approaches to deciding properties in logic in general, and in particular in description logics; namely, tableau-based and automata-based decision procedures. Several logics will be used as a showcase to shine light of the peculiarities of each of these methods. First we introduce the main reasoning problems for members of the family of Description Logics having different expressivity, for which we will present tableau-based decision procedures. These will work as a basis from which our general notions of tableaux (Chapter 3) and blocking (Chapter 4) will be constructed. For the most expressive Description Logic presented in this work, that is, $\mathcal{ALC}$ with $\mathcal{SI}$ TBoxes, we introduce also an automata-based decision procedure that relies on the fact that this logic has the *tree-model property* by constructing representations of all the tree-shaped models. As an example of an automata-based decision procedure requiring additional acceptance conditions, we include the problem of deciding satisfiability of Linear Temporal Logic formulae. The use of this logic to exemplify our automata-based approach is further motivated by the fact that automata-based decision procedures have been successfully applied in practice for program verification [Var96] or model checking.

In the first two sections of this chapter we describe the logics under consideration: we first give a brief introduction to Description Logics and their main reasoning problems in Section 2.1, followed by an introduction to Linear Temporal Logic. Then, in Section 2.3, we present tableau-based decision algorithms for the problems relevant to Description Logics. Finally, the automata-based decision procedures are described in Section 2.4.

2.1 Description Logics

Description Logics (DLs) [BCM$^+$03] are a family of logic-based knowledge representation formalisms commonly used to represent the knowledge of a given application domain in a structured manner which is also easy to understand. The main feature relating all the logics in this family is the use of *concepts* that intuitively describe properties held by individuals in a domain, and *roles*, or relations between two such

individuals. What differentiates one DL from another is the constructors it uses for generating complex concepts and roles from a set of primitive ones, also called *concept-* and *role-names*. The choice of these constructors obviously has an impact not only on the expressivity of the logic, but also on the complexity of its reasoning problems.

The most basic constructors are the Boolean ones; that is, disjunction, conjunction and negation – denoted as $\sqcup, \sqcap$, and $\neg$, respectively – with the same intended meaning as their propositional logic counterparts. The quantifiers $\forall$ and $\exists$ allow us to jump beyond the realm of propositional logic and reason about the relations between individuals, each satisfying a given property. The *value restriction* $\forall r.C$ is satisfied by each individual x such that, if x is related to another individual y via the role r, then y satisfies the concept C. Likewise, the *existential restriction* $\exists r.C$ describes the individuals that are related via r to some individual belonging to C. One can additionally use the *top* $\top$ and the *bottom* $\bot$ concepts, that are satisfied by all and none individuals, respectively. The most basic DL using all of these constructors is $\mathcal{ALC}$, an acronym that stands for *attributive language with complements* originally introduced in [SS91].

Definition 2.1 (Syntax of $\mathcal{ALC}$). *Let* CN *and* RN *be two disjoint sets of* concept- *and* role-names*, respectively. The set of* $\mathcal{ALC}$ concept terms *is the smallest set containing* CN *and such that if* C, D *are two concept terms and* $r \in \mathsf{RN}$ *is a role name, then* $C \sqcap D, C \sqcup D, \neg C, \exists r.C$ *and* $\forall r.C$ *are all concept terms.* ■

If it is clear from the context we will usually say $\mathcal{ALC}$ *concept* or even just *concept* instead of using the longer name "$\mathcal{ALC}$ concept term".

Let us instantiate Definition 2.1 with an example. When modeling the domain of human evolution, one can describe a descendant of an *Homo ergaster* with the concept $\exists$has-ancestor.HErgaster, or a species whose evolutionary descendants belong all to the genus *Homo* using the concept $\forall$has-descendant.Homo.

In addition to the constructors used by this logic, several others have been considered in the DL literature such as (qualified or unqualified) number restrictions, nominals, and role compositions, among others (see [BCM+03]). For the scope of this work we will focus on the DL $\mathcal{ALC}$ as well as on the logic $\mathcal{HL}$, which is the sub-logic of $\mathcal{ALC}$ that allows only for conjunction and the top concept as a constructors. The main decision problems for these two logics and different sets of axioms will require the introduction of several distinct techniques for solving them. These techniques will then be formalised when defining general decision procedures and restrictions in the following chapters.

Representing the knowledge of a domain may require the use of specific individuals that can act as instances of concept terms. These individuals may receive any name in the formal description, but must be interpreted as elements of the domain. For this reason, we will use an additional set IN of *individual names* disjoint from both CN and RN.

Returning to our evolutionary example, we may introduce the individual name Lucy whose intuitive task is to represent the famous *Australopithecus afarensis* fossil.

The importance of DLs as a knowledge representation formalism relies on their

formal semantics based on interpretations that map all concept- and role-names to sets of individuals and sets of pairs of individuals of a specific domain, respectively.

Definition 2.2 (Semantics of $\mathcal{ALC}$). *Let* CN, RN *and* IN *be pairwise disjoint sets of concept-, role- and individual names, respectively. An* interpretation *is a tuple of the form* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$*, where* Δ *is a set, called the* domain *of* $\mathcal{I}$*, and* $\cdot^{\mathcal{I}}$ *is a function mapping every individual name* $a \in \mathsf{IN}$ *to an element* $a^{\mathcal{I}} \in \Delta$*, every concept name* $A \in \mathsf{CN}$ *to a subset* $A^{\mathcal{I}} \subseteq \Delta$ *and every role name* $r \in \mathsf{RN}$ *to a set of pairs* $r^{\mathcal{I}} \subseteq \Delta \times \Delta$*.*

The function $\cdot^{\mathcal{I}}$ *is inductively extended to all concept terms as follows:*

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$;
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$;
- $(\neg C)^{\mathcal{I}} = \Delta \setminus C^{\mathcal{I}}$;
- $(\exists r.C)^{\mathcal{I}} = \{d \in \Delta \mid$ *there is an* e *such that* $(d, e) \in r^{\mathcal{I}}$ *and* $e \in C^{\mathcal{I}}\}$;
- $(\forall r.C)^{\mathcal{I}} = \{d \in \Delta \mid$ *for all* e*, if* $(d, e) \in r^{\mathcal{I}}$*, then* $e \in C^{\mathcal{I}}\}$.

■

The domain knowledge is stored using a set of axioms that restrict the set of admissible interpretations by imposing conditions on the concepts (terminological axioms), individuals (assertional axioms), or roles (role axioms). We distinguish two kinds of terminological axioms: *concept definitions* that, with some appropriate syntactic restrictions, help to define macros or abbreviations of concept terms, and *general concept inclusion axioms* that express an inclusion relation between two concepts.

Definition 2.3 (Terminological axiom, (Acyclic, General) TBox). *A* concept definition *is of the form* $A \doteq C$*, where* $A \in \mathsf{CN}$ *is a concept name and* C *is a concept term. A* general concept inclusion axiom *(or* GCI *for short) is an expression of the form* $C \sqsubseteq D$ *with* C *and* D *concept terms.*

An acyclic TBox *is a set* $\mathcal{T}$ *of concept definitions that satisfies the following conditions:*

- *for every concept name* A*, there is at most one concept definition in* $\mathcal{T}$ *of the form* $A \doteq C$*; and*
- *there is no sequence of concept definitions* $A_1 \doteq C_1, A_2 \doteq C_2, \ldots A_n \doteq C_n$ *such that for every* $1 < j \leq n$*,* A_j *appears in* C_{j-1} *and* A_1 *appears in* C_n*.*

A general TBox *is simply a set of GCIs.* ■

Intuitively, the conditions imposed on acyclic TBoxes ensure that every concept name is defined only once, and the right-hand-side of each definition has no direct or indirect reference to its left-hand-side. General TBoxes are indeed more general than acyclic ones, in the first place because each concept definition $A \doteq C$ can be represented with the GCIs $A \sqsubseteq C, C \sqsubseteq A$, and second because there is no restriction

on the left-hand-side elements appearing on the right-hand-side concept term of a GCI.

For example, we can define our species, *Homo sapiens*, in terms of its evolutionary ancestors and siblings using the concept definition

$$\mathsf{HSapiens} \doteq \exists \mathsf{has\text{-}ancestor}.\mathsf{HHeidelburgensis} \sqcap \neg \mathsf{HNeanderthalensis}.^{6}$$

We can also express that *Homo* and *Australopithecus* are two disjoint genera, i.e., that no individual can belong to both of them, with the GCI $\mathsf{Homo} \sqcap \mathsf{Australopithecus} \sqsubseteq \bot$.

Notice that the restrictions imposed in an acyclic TBox ensure that each concept definition actually acts as a definition of the concept name appearing in its left-hand side as an abbreviation of the (complex) concept term in its right-hand side. In particular, this means that acyclic TBoxes do not add any expressive power to the language. Nonetheless, they allow us to express complex concept terms and reason about them in a more succinct fashion [Neb90, Lut99].

In some cases, restricting the concepts does not suffice to fully represent the knowledge domain, and we want to specify some individuals as members of specific concept terms. For instance, in the evolutionary ontology we need to express that Lucy is an *Australopithecus afarensis*. This fact can be represented by the so-called *assertional axiom* $\mathsf{AAfarensis}(\textsc{Lucy})$.

Definition 2.4 (Assertional axiom, ABox). *An* assertional axiom *is an expression of the form $C(a)$, or $r(a, b)$ where $a, b \in \mathsf{IN}$ are individual names, C is a concept term, and r is a role name. A set of assertional axioms is called an* ABox. ■

In the same way that we restricted the relations between concept terms by means of terminological axioms, we can limit the possible interpretations of the roles used in their construction by imposing a set of role axioms. As in the case of the constructors for concept terms, several distinct role axioms have been considered in the literature [HS04, HKS05, HKS06]. In the present work we will focus solely on axioms that force roles to be transitive or inverses of each other.

Definition 2.5 (Role axiom, $\mathcal{SI}$-TBox). *Let $r, s \in \mathsf{RN}$ be two distinct role names. The expressions $\mathsf{trans}(r)$ and $\mathsf{inv}(r, s)$ denote a* transitivity- *and* inverse axiom*, respectively. A* role axiom *is either a transitivity- or an inverse axiom.*

An (acyclic, general) $\mathcal{SI}$-TBox *is a set $S = T \cup R$ where T is an (acyclic, general) TBox and R is a set of role axioms such that every $r \in \mathsf{RN}$ appears in at most one inverse axiom.*[7] ■

[6] Unfortunately, there is at the present no full consensus on the evolutionary history of human-kind. The examples presented here show only one of the most accepted views, and are are intended only as illustrations for our definitions.

[7] The DL $\mathcal{ALC}$ extended with transitive and inverse roles, called $\mathcal{SI}$ in the DL literature, is usually defined in a different manner, using an inverse *constructor* instead of axioms restricting the interpretation of the role. We decided to use the equivalent axiomatic restriction since an incorrect use of inverses may lead to unsatisfiability, and we want to be able to detect this cause when performing pinpointing.

Syntax	Semantics
$A \doteq C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
$\mathsf{trans}(r)$	$r^{\mathcal{I}}$ is transitive
$\mathsf{inv}(r, s)$	$r^{\mathcal{I}}$ is the inverse of $s^{\mathcal{I}}$

Figure 2.1: Semantics of axioms

Once again using the evolutionary ontology as an example, the role has-descendant should be interpreted as being transitive, which can be enforced by including the axiom trans(has-descendant), and as being the inverse role of has-ancestor, which is easily done with the introduction of the role axiom inv(has-ancestor, has-descendant).

When axioms are used, the semantics of $\mathcal{ALC}$ and $\mathcal{HL}$ concepts are restricted to consider only those interpretations that satisfy the restrictions imposed by the specified axioms. Such interpretations are called *models*. In the presence of axioms, not all interpretations are taken into account, but only those that *model* them. In other words, only those interpretations that satisfy the semantic restrictions imposed by the axioms, as summarized in Figure 2.1, are rendered relevant.

Definition 2.6 (Semantics of axioms). *Given a set of axioms T, $\mathcal{I}$ is a* model of *T iff for every axiom $t \in T$, $\mathcal{I}$ satisfies the semantics of t as shown in Figure 2.1.* ■

The first question that can be asked of a set of axioms is whether it is *consistent*; that is, whether it is possible to construct a model for it or not. This question is typically interesting in the presence of assertional axioms since we want to know whether some specific individuals may satisfy the restrictions we are imposing on them. Additionally to consistency, two of the main decision problems that arise in DLs are the *satisfiability* and *subsumption* problems. The satisfiability problem consists in checking whether there exist a model for a given set of axioms that maps a given concept term to a non-empty set. On the other hand, the subsumption problem checks whether every model interprets a concept as a subset of another concept. A more formal definition follows.

Definition 2.7 (Consistency, satisfiability, subsumption). *Let T be a set of axioms and C, D two concept terms. We say that T is* consistent *iff there is a model of T. C is* satisfiable w.r.t. *T iff there exists a model $\mathcal{I}$ of T such that $C^{\mathcal{I}} \neq \emptyset$. C is* subsumed by *D* w.r.t. *T (denoted $C \sqsubseteq_T D$) iff for every model $\mathcal{I}$ of T it holds that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.* ■

It is worth noticing that, in the presence of the negation constructor, these last two problems are polynomially reducible to each other. On one hand, a concept C is satisfiable w.r.t. T iff $C \not\sqsubseteq_T \bot$; conversely, $C \sqsubseteq_T D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. T. For this reason, it suffices to design an algorithm that decides any of those problems in order to solve the other. In this work, we will focus on the satisfiability problem when dealing with the DL $\mathcal{ALC}$. In the case of the very inexpressive logic

$\mathcal{HL}$, there are no means for expressing negation, and hence all concepts described in it are always satisfiable. For that reason, we will focus on the subsumption problem when reasoning in this logic. It is also relevant to realise that deciding satisfiability of a concept C w.r.t. a set of axioms T is equivalent to deciding consistency of the set $T \cup \{C(a)\}$ where a is an individual name not appearing in T. Basically, since C is satisfiable w.r.t. T iff there is a model that maps C to a non-empty set, we can force the interpretation of C to contain a random element in the domain.

Later on in this chapter we will describe well known algorithms for solving subsumption of $\mathcal{HL}$-concepts w.r.t. TBoxes, and satisfiability of $\mathcal{ALC}$ concepts w.r.t. to the distinct kinds of standard sets of axioms, with an emphasis on the characteristics that are shared between them, and the specific elements that differentiate each particular case. Before that, we will introduce Linear Temporal Logic with its relevant decision problem.

2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) is an extension of Propositional Logic that allows reasoning about temporal properties, where time is seen as discrete and linear [GPSS80, Pnu77]. The syntax of this logic extends the usual propositional constructors with the constructors *next*, denoted as $\bigcirc$, and *until*, denoted as $\mathcal{U}$. Intuitively, the formula $\bigcirc\phi$ expresses that the formula ϕ must be true in the next point in time, while $\phi\mathcal{U}\psi$ is true if there is a moment in the future where ψ is true, and ϕ is true at every moment between the present and the one satisfying ψ. We will now formalise these notions.

Definition 2.8 (Syntax of LTL). *Let* P *be a set of propositional variables. The set of* LTL formulae *is the smallest set where*

- *all propositional variables are LTL formulae;*
- *if* ϕ *and* ψ *are LTL formulae, then so are* $\neg\phi, \phi \wedge \psi, \bigcirc\phi$ *and* $\phi\mathcal{U}\psi$.

■

The semantics of this logic use the notion of *computations*, which intuitively correspond to interpretations, as defined for DLs, but where the domain is fixed to be the set of natural numbers.

Definition 2.9 (Semantics of LTL). *A* computation *is a function* $\pi : \mathbb{N} \to \mathscr{P}(\mathsf{P})$, *where* $\mathbb{N}$ *represents the set of natural numbers. This function* π *is extended to LTL formulae as follows, for every* $i \in \mathbb{N}$*:*

- $\neg\phi \in \pi(i)$ *iff* $\phi \notin \pi(i)$; $\phi \wedge \psi \in \pi(i)$ *iff* $\{\phi, \psi\} \in \pi(i)$;
- $\bigcirc\phi \in \pi(i)$ *iff* $\phi \in \pi(i+1)$; *and*
- $\phi\mathcal{U}\psi \in \pi(i)$ *iff there is a* $j \geq i$ *such that* $\psi \in \pi(j)$ *and for all* $k, i \leq k < j$ *it holds that* $\phi \in \pi(k)$.

An LTL formula ϕ is satisfiable *if there is a computation π such that $\phi \in \pi(0)$.* ■

One is usually interested in deciding whether a given LTL formula is satisfiable or not. Since the main goal of this work is related with reasoning with respect to sets of axioms, we will be interested in a variation of the satisfiability problem, where LTL formulae are used as axioms. Given a set of LTL formulae $\mathcal{R}$, we consider the problem of deciding whether the conjunction of all formulae in $\mathcal{R}$ is satisfiable or not. If this conjunction is unsatisfiable, pinpointing will allow us to detect the subsets of formulae, i.e., the conjuncts, responsible for this. We will further assume that there is a fixed conjunct that is always present. In summary, our problem receives as input a static LTL formula ϕ and a set of refutable LTL formulae $\mathcal{R}$, and decides whether the conjunction of all these formulae is satisfiable or not. We now formally define this problem, which we will call *axiomatic satisfiability.*

Definition 2.10 (Axiomatic satisfiability). *Let ϕ be an LTL formula and $\mathcal{R}$ a set of LTL formulae. We say that ϕ is* axiomatic satisfiable *w.r.t. $\mathcal{R}$ if there is a computation π such that $\mathcal{R} \cup \{\phi\} \subseteq \pi(0)$.*[8] *In this case, π is called a* computation for *$(\phi, \mathcal{R})$.* ■

At the end of this chapter we will present a procedure based on Büchi automata that will allow us to correctly decide axiomatic satisfiability of LTL formulae.

Depending on the characteristics of the logic in use and the kind of axioms considered, distinct methods can be applied to solve its decision problems. In description logics, the two prominent approaches for deciding consistency, subsumption, or satisfiability of concept terms are the tableau-based and the automata-based methods. In the case of consistency or satisfiability of concept terms, the tableaux-based approach tries to construct a model in a top-down (usually non-deterministic) fashion, until the model is finished or it is clear that no adequate interpretation can exist. The models created this way usually have an underlying tree-shape. For that reason, whenever the logic in consideration does not have the *finite* tree model property (as is the case of $\mathcal{ALC}$ with general TBoxes) additional restrictions need to be specified to stop the execution in finite time while retaining the correctness of the method. As we will see later in Section 2.3, in these infinite models it is possible to find a pattern that repeats after a finite number of nodes. Thus, only finite information is necessary to reproduce the infinite model. The idea of stopping the execution of the tableau once an appropriate pattern has been found receives the name of *blocking* in the DL literature.

The automata-based approach is usually more straightforward. The idea consists in constructing an automaton that accepts exactly all those tree-shaped models. The deterministic and polynomial-time emptiness test on this automaton yields the desired decision. In DLs, the runs accepted by such an automaton are in fact infinite tree models, where every node represents an individual. The nodes are then labeled with the concepts that they satisfy within the interpretation with the help of the transition relation of the automaton. The decision procedure for axiomatic satisfiability of LTL formulae follows a similar idea, constructing an automaton whose accepted runs consist

[8] Notice that this is equivalent to saying that $\phi \wedge \bigwedge_{\psi \in \mathcal{R}} \psi$ is satisfiable.

on the computations for the input. Given the nature of the *until* operator, whose satisfiability can be delayed as much as desired within the computation, it is necessary to use an acceptance condition that ensures that this delay is not performed forever, but every until formulae is eventually satisfied.

2.3 Tableau-Based Decision Algorithms

We proceed now to present several decision procedures that exemplify the main ideas of tableaux briefly mentioned above. We first present a deterministic algorithm that decides subsumption in $\mathcal{HL}$ with general TBoxes. This algorithm has the benefits of being deterministic and running in polynomial time, and hence allowing us to detect the increase of complexity caused by trying to explain the subsumptions that hold, compared with merely detecting if they follow from the general TBox or not (see Chapter 6). We continue with a description of the tableau-based algorithms for deciding consistency of ABoxes and then satisfiability of $\mathcal{ALC}$ concepts w.r.t. acyclic, general, and $\mathcal{SI}$-TBoxes incrementally: we re-use the consistency algorithm for ABoxes to decide satisfiability, by simply adding a series of expansion rules that deal with the axioms being considered.

The algorithm for $\mathcal{HL}$ is a special case of the subsumption algorithm for the DL $\mathcal{EL}$ that also runs in polynomial time [Baa03a, Baa03b].[9] The other tableau methods are well known algorithms. For a deeper description, including more expressive constructors not treated here, such as number restrictions and role hierarchies, and complexity and run-time analysis of these methods, refer to [BS01].

2.3.1 Subsumption in $\mathcal{HL}$ with General TBoxes

Recall that in $\mathcal{HL}$, all concept terms consist of conjunctions of concept names, and thus all GCIs in this logic are of the form

$$\mathbf{A}_1 \sqcap \mathbf{A}_2 \sqcap \ldots \sqcap \mathbf{A}_n \sqsubseteq \mathbf{B}_1 \sqcap \mathbf{B}_2 \sqcap \ldots \sqcap \mathbf{B}_m$$

where $n, m \geq 0$ and each $\mathbf{A}_i$ and $\mathbf{B}_i$ is a concept name in CN. Intuitively, an axiom of this form states that if a concept is subsumed by *all* the concepts $\mathbf{A}_1 \ldots \mathbf{A}_n$, then it is also subsumed by each and every one of the concepts $\mathbf{B}_1 \ldots \mathbf{B}_m$. Our algorithm will iteratively make such knowledge explicit based on the explicit subsumption relations known so far. This information will be stored in a set $\mathcal{A}$ of pairs of the form $(\mathbf{A}, \mathbf{B})$, where $\mathbf{A}$ and $\mathbf{B}$ are concept names, with the intended meaning that $(\mathbf{A}, \mathbf{B})$ is present if and only if $\mathbf{B}$ subsumes $\mathbf{A}$.

The algorithm starts with the trivial knowledge stating that every concept name appearing in the general TBox $\mathcal{T}$ is subsumed by itself; i.e., it initialises the set $\mathcal{A}$ with $\mathcal{A} = \{(\mathbf{A}, \mathbf{A}) \mid \mathbf{A} \in \mathsf{CN} \text{ appears in } \mathcal{T}\}$, and then repeatedly applies the expansion rule hl that is shown in Figure 2.2.

Obviously, in order to ensure termination of this expansion method, the rule hl should only be applied if its application will result in a real expansion of the set $\mathcal{A}$,

[9] $\mathcal{EL}$ is the superlogic of $\mathcal{HL}$ that allows also for existential restrictions.

hl	if $\prod_{i=1}^{n} \mathbf{A}_i \sqsubseteq \prod_{j=1}^{m} \mathbf{B}_j \in \mathcal{T}$ and $\{(\mathbf{A}, \mathbf{A}_i) \mid 1 \leq i \leq n\} \subseteq \mathcal{A}$, then add $(\mathbf{A}, \mathbf{B}_j)$ to $\mathcal{A}$ for all $1 \leq j \leq m$.

Figure 2.2: Expansion rule for deciding subsumption in $\mathcal{HL}$

that is, if there is at least one j such that $(\mathbf{A}, \mathbf{B}_j) \notin \mathcal{A}$. Otherwise, we could loop indefinitely applying the same rule once and again without achieving any progress. Given this restriction, it is clear that the expansion rule is applied at most once for each GCI and concept name in $\mathcal{T}$. Thus, the algorithm finishes in polynomial time measured on the size of the TBox. When no more pairs can be added to $\mathcal{A}$ by an application of this rule, it is the case that $(\mathbf{A}, \mathbf{B}) \in \mathcal{A}$ iff $\mathbf{A} \sqsubseteq_{\mathcal{T}} \mathbf{B}$, for all concept names $\mathbf{A}, \mathbf{B}$ appearing in $\mathcal{T}$. As it was said before, this algorithm is in essence a special case of the subsumption algorithm for $\mathcal{EL}$. For a proof of correctness and its polynomial execution time, refer to [BBL05, Bra04a].

2.3.2 Consistency of $\mathcal{ALC}$ ABoxes

We move now beyond $\mathcal{HL}$ to the more expressive logic $\mathcal{ALC}$, and consider first the problem of consistency of an ABox. This problem corresponds to deciding whether there is a model for a given set of assertional axioms. In order to solve it, we begin by stating all the restrictions imposed by the axioms in the input and then expand this knowledge according to the semantics of the constructors used (see Definition 2.2). When this expansion process terminates, we either have a model (and hence the ABox is consistent) or there is an obvious contradiction. Actually, due to the presence of disjunction, this process has a (do not know) non-deterministic factor, and possibly several model candidates would have to be tried. Each model candidate will be represented as a set $\mathcal{A}_i$ of assertions of the form $C(a)$ or $r(a, b)$, where C is a concept term, r is a role name, and a and b are individuals. In other words, we use ABoxes also to represent (partial) models. To deal with the non-determinism, we consider all these ABoxes simultaneously, as elements of a set $\mathcal{M}$, rather than only one at a time. This can be thought of as testing all the possible model candidates concurrently.

The algorithm starts with the only model candidate consisting of the input ABox $\mathcal{A}_0$; that is, it initialises $\mathcal{M} = \{\mathcal{A}_0\}$. This set is then modified by successive applications of the expansion rules shown in Figure 2.3, where a rule is applied to one set $\mathcal{A}$ in $\mathcal{M}$ at a time. These rules are applied until none of them can be applied anymore, extending the set $\mathcal{M}$ of model candidates. An ABox $\mathcal{A} \in \mathcal{M}$ is said to have a *clash* if there is an individual name x occurring in $\mathcal{A}$ and a concept name $\mathbf{A}$ such that $\{\mathbf{A}(x), \neg\mathbf{A}(x)\} \subseteq \mathcal{A}$.

This expansion process is guaranteed to finish after a finite number of rule applications, and when it does so, the resulting set $\mathcal{M}$ is such that the original ABox $\mathcal{A}_0$ is consistent if and only if there is a model candidate $\mathcal{A} \in \mathcal{M}$ that does not have any clash [BH91, Hol96].

Recall, from the definition of satisfiability, that a concept is satisfiable with respect

$\mathsf{alc}_{\sqcap}$	if $(C \sqcap D)(x) \in \mathcal{A}$ but $\{C(x), D(x)\} \not\subseteq \mathcal{A}$, then replace $\mathcal{A}$ by $\mathcal{A}' = \mathcal{A} \cup \{C(x), D(x)\}$.
$\mathsf{alc}_{\sqcup}$	if $(C \sqcup D)(x) \in \mathcal{A}$ but $\{C(x), D(x)\} \cap \mathcal{A} = \emptyset$, then replace $\mathcal{A}$ by the two sets $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}$ and $\mathcal{A}'' = \mathcal{A} \cup \{D(x)\}$.
$\mathsf{alc}_{\forall}$	if $\{(\forall r.C)(x), r(x,y)\} \subseteq \mathcal{A}$ but $C(y) \notin \mathcal{A}$, then replace $\mathcal{A}$ by $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$.
$\mathsf{alc}_{\exists}$	if $(\exists r.C)(x) \in \mathcal{A}$ but there is no individual name z such that $\{r(x,z), C(z)\} \subseteq \mathcal{A}$, then replace $\mathcal{A}$ by $\mathcal{A}' = \mathcal{A} \cup \{C(y), r(x,y)\}$ where y is an individual name not occurring in $\mathcal{A}$.

Figure 2.3: Expansion rules for the tableau algorithm for consistency of $\mathcal{ALC}$ ABoxes

$\mathsf{alc}_{\doteq}$	if $A(x) \in \mathcal{A}$ and $A \doteq C \in \mathcal{T}$ but $C(x) \notin \mathcal{A}$, then replace $\mathcal{A}$ by $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}$.

Figure 2.4: Rule $\mathsf{alc}_{\doteq}$ for deciding satisfiability of $\mathcal{ALC}$ concepts w.r.t. acyclic TBoxes

to a given TBox if and only if there is a model that maps it to a non-empty set. In other words, the concept C is satisfiable w.r.t. $\mathcal{T}$ iff the ABox $\{C(a)\}$ is consistent (w.r.t. $\mathcal{T}$), where a is an arbitrary individual name. If we consider an empty TBox, then the consistency algorithm described in this subsection would yield the desired decision procedure. In general, nonetheless, we require to extend it to deal with the terminological axioms. The following subsections deal with this.

2.3.3 Satisfiability of $\mathcal{ALC}$ Concepts with Acyclic TBoxes

As noticed before, acyclic TBoxes work basically as abbreviations of more complex concept terms and do not add to the expressivity of $\mathcal{ALC}$. In fact, reasoning with respect to an acyclic TBox can be reduced to reasoning with an empty TBox by a process known as *unfolding*: replacing, for every concept definition $A \doteq C$, every occurrence of the concept name A by its defined concept C. Unfortunately, this reduction may produce a concept that is exponential in the size of the original TBox (see [Neb90] for an example supporting this claim).

In the DL $\mathcal{ALC}$, one can avoid this exponential blow-up by only unfolding at the moments where it is necessary to explore deeper in a concept definition [Lut99]. This method, commonly referred to as *lazy unfolding* can be easily implemented in our tableau system for deciding satisfiability of $\mathcal{ALC}$ concept terms, by simply adding the rule $\mathsf{alc}_{\doteq}$ (shown in Figure 2.4) to the tableau for ABox consistency (Figure 2.3).

The procedure works exactly in the same fashion as the one described in the pre-

$\mathsf{alc}_{\sqsubseteq}$	if x is an individual name appearing in $\mathcal{A}$ but $(\neg C \sqcup D)(x) \notin \mathcal{A}$, then replace $\mathcal{A}$ by $\mathcal{A}' = \mathcal{A} \cup \{(\neg C \sqcup D)(x)\}$.

Figure 2.5: Rule $\mathsf{alc}_{\sqsubseteq}$ for reasoning with GCIs

vious subsection: it starts with the unique model candidate having only the assertion $C(a)$ where C is the concept being tested for satisfiability and a an arbitrary individual name. It then repeatedly applies the expansion rules until none is applicable anymore. It can be easily shown that this process finishes after a finite number of rule applications, at which point it holds that C is satisfiable if and only if there is a model candidate that does not contain any clash [Lut99].

2.3.4 Satisfiability of $\mathcal{ALC}$ Concepts with General TBoxes

When dealing with general concept inclusion axioms, we can no longer assume that the TBox defines abbreviations of more complex concepts, which means that the idea of lazy unfolding is no longer applicable. It is thus necessary to implement a different method that can deal with this kind of terminologies. An analysis of the semantics of the axioms that constitute general TBoxes reveals that they express a restriction on the concepts to which every individual name must belong. More clearly, a GCI of the form $C \sqsubseteq D$ expresses that every individual that belongs to the concept C must also belong to D. We can also express this by forcing every individual to either not belong to C, or otherwise belong to D. In other words, for every individual name a, $(\neg C \sqcup D)(a)$ must hold. The rule $\mathsf{alc}_{\sqsubseteq}$ shown in Figure 2.5 implements this idea, forcing every individual name used in the ABox working as a model candidate to satisfy each of the restrictions imposed by the GCIs. These rules are applied in the same fashion as in the previous subsections, starting with only a model candidate stating the non-emptiness of the interpretation of the concept being tested. More formally, we begin with the set $\mathcal{M} = \{\{C(a)\}\}$ where C is the concept being tested for satisfiability, and a an arbitrary individual name. We then apply the expansion rules in any order. Unfortunately, and contrary to the previous methods presented so far, application of this set of rules is not guaranteed to finish after a finite number of steps, as shown in the following example.

Example 2.11. *Consider the TBox $\mathcal{T}$ containing only one axiom $\mathcal{T} = \{A \sqsubseteq \exists r.A\}$. If we want to test for satisfiability of the concept A, then the tableau algorithm described here will start with $\mathcal{M} = \{\mathcal{A}_0\}$, where $\mathcal{A}_0 = \{A(a_0)\}$. At this point, only the rule $\mathsf{alc}_{\sqsubseteq}$ is applicable to the only model candidate present in $\mathcal{M}$. Its application replaces $\mathcal{A}_0$ with $\mathcal{A}_1 = \mathcal{A}_0 \cup \{(\neg A \sqcup \exists r.A)(a_0)\}$. Again, only one rule is applicable, which is the $\mathsf{alc}_{\sqcup}$ rule. Its application replaces $\mathcal{A}_1$ with the two sets $\mathcal{A}_2 = \mathcal{A}_1 \cup \{(\exists r.A)(a_0)\}$ and $\mathcal{A}'_2 = \mathcal{A}_1 \cup \{\neg A(a_0)\}$. Notice that no rule is applicable to $\mathcal{A}'_2$, and that it contains a clash, namely $A(a_0), \neg A(a_0)$. On the other hand, the rule $\mathsf{alc}_{\exists}$ is applicable to $\mathcal{A}_2$ whose application substitutes that model candidate with $\mathcal{A}_3 = \mathcal{A}_2 \cup \{r(a_0, a_1), A(a_1)\}$. It is easy to see that the same sequence of rule applications is possible, leading to a model candidate having the assertion $A(a_2)$ where a_2 is a new individual name, and*

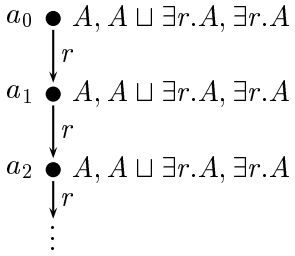

Figure 2.6: An infinite model

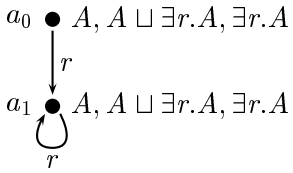

Figure 2.7: A finite *equivalent* model

hence the same sequence of rule applications is once again possible. This leads to a non-terminating sequence of rule applications. ■

From this example we know that the algorithm is not ensured to terminate after a finite number of rule applications. Nonetheless, if we allowed the process to run indefinitely, we would notice that all the individuals used in the infinite model constructed this way satisfy the same concepts (see Figure 2.6). In that sense, one can say that the algorithm has been trapped in a cycle. Furthermore, we notice that an infinite expansion is only possible by the addition of new individual names; that is, by the use of so-called *generating rules*. In the present case, the only generating rule is $\mathsf{alc}_\exists$. To regain termination, we need then to device a mechanism that detects when the expansion has found a cycle and then avoids generating new individuals by reusing the information of the cycle. This mechanism is called *blocking* in the DL literature [BS01].

The blocking mechanism for $\mathcal{ALC}$ w.r.t. general TBoxes is based on the individual names used in the model candidate. We say that an individual name x is *blocked* by the individual name y if y appears in all the assertions in which x appears; more formally, if $\{D \mid D(x) \in \mathcal{A}\} \subseteq \{D \mid D(y) \in \mathcal{A}\}$. If an individual x is blocked by y, then the rule $\mathsf{alc}_\exists$ is not applied when triggered by an assertion of the form $\exists r.C(x)$. As the concepts satisfied by a blocked node form a subset of those satisfied by the blocking node, this particular instance receives the name of *subset blocking*. Intuitively, a blocked individual x should be able to reuse the role successors of y instead for generating new ones that will have the same shape. In our example, we could have avoided generating the new individual a_2 by simply reusing the successor a_1 of a_0 as the new successor of a_1 (see Figure 2.7).

In order for this idea to work correctly, we need to restrict the set of individual names that are able to block a given individual. Basically, it is necessary to avoid a situation in which a pair of nodes are blocking each other, which would produce an early termination of the algorithm that might destroy its soundness. For the algorithm in hand, it is enough to force the blocking node to be a predecessor of the blocked node. The infinite tree-shaped model can be recovered from the model obtained from blocking by an *unraveling* process that creates new successors for those nodes pointing *backwards* in the tree-like model. This tableau algorithm, with the use of subset blocking, is always terminating and decides satisfiability of a concept w.r.t. a general TBox in the same way as the one described in the previous sections: C is

alc_+	if $\{(\forall r.C)(x), r(x,y)\} \subseteq \mathcal{A}$ and $\mathsf{trans}(r) \in \mathcal{T}$ but $(\forall r.C)(y) \notin \mathcal{A}$, then replace $\mathcal{A}$ by $\mathcal{A}' = \mathcal{A} \cup \{(\forall r.C)(y)\}$.
alc_-	if $\{(\forall r.C)(y), s(x,y)\} \subseteq \mathcal{A}$ and $\{\mathsf{inv}(r,s), \mathsf{inv}(s,r)\} \cap \mathcal{T} \neq \emptyset$ but $C(x) \notin \mathcal{A}$, then replace $\mathcal{A}$ by $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}$.
$\mathsf{alc}_\pm$	if $\{(\forall r.C)(y), s(x,y)\} \subseteq \mathcal{A}$ and $\{\mathsf{inv}(r,s), \mathsf{inv}(s,r)\} \cap \mathcal{T} \neq \emptyset$ and $\{\mathsf{trans}(r), \mathsf{trans}(s)\} \cap \mathcal{T} \neq \emptyset$ but $(\forall r.C)(x) \notin \mathcal{A}$, then replace $\mathcal{A}$ by $\mathcal{A}' = \mathcal{A} \cup \{(\forall r.C)(x)\}$.

Figure 2.8: Rules for dealing with transitivity and inverse axioms

satisfiable if and only if the algorithm starting with $\{\{C(a)\}\}$ yields a model candidate that has no clash [BDS93].

2.3.5 Satisfiability of $\mathcal{ALC}$ Concepts with $\mathcal{SI}$-TBoxes

Once we introduce inverse and transitivity axioms, the decision procedure becomes more complex. To deal with transitivity, it is helpful to notice that the only semantical influence of these axioms on the construction of a model is with respect to the universal restrictions. If r is a transitive role, then a universal restriction imposed in an individual x needs to be satisfied not only by its direct r successors, but also by their own r successors and so on. Clearly, we can perform this task with the help of a tableau rule. The rule alc_+ in Figure 2.8, analogous to the one introduced in [Hor98] for dealing with transitive roles, shows exactly this behaviour.

Inverse axioms need a similar approach. When an inverse axiom is present, the restrictions may need to be propagated *backwards* along the inverse roles. In other words, if we have $r(x,y)$ and $(\forall s.C)(y)$, where $\mathsf{inv}(r,s)$, then we should be able to deduce $C(x)$. Rule alc_-, shown in Figure 2.8, deals with this fact.

One has to notice still that if a role is transitive, then its inverse must also be transitive. For that reason, whenever a role appears both in a transitivity and an inverse axiom, we should be able to combine the propagation of universal restrictions due to transitivity with the *backwards* propagation due to inverses. Hence, we introduce the rule $\mathsf{alc}_\pm$ to the tableaux algorithm dealing with this logic.

Depending on whether we have an acyclic or a general $\mathcal{SI}$ TBox, we need to use the rule $\mathsf{alc}_\doteq$ or $\mathsf{alc}_\sqsubseteq$, accordingly, in addition to the rules presented here to deal with the rest of the axioms appearing in it. Obviously, the rules depicted in Figure 2.3 are also necessary.

The presence of transitive axioms leads to a non-terminating tableau algorithm, even in the case of acyclic $\mathcal{SI}$-TBoxes. Hence, we require an appropriate blocking condition that ensures termination after a finite number of rule applications. Unfortunately, due to inverse axioms, we cannot use subset blocking as presented in the previous subsection. This is shown in the following example.

Figure 2.9: Failure of subset blocking with $\mathcal{SI}$-TBoxes

Example 2.12 (Failure of subset blocking). *Consider the situation shown in the left part of Figure 2.9, where we are testing for satisfiability of the concept A w.r.t. the general $\mathcal{SI}$-TBox $\mathcal{T} = \{A \sqsubseteq \exists s.(\forall r.\forall r.\neg B \sqcap B \sqcap \exists s.A), \mathsf{inv}(r, s)\}$. For brevity, the figure does not show all the concepts obtained by application of the $\mathsf{alc}_{\sqsubseteq}$ rule and the subsequent expansion by $\mathsf{alc}_{\sqcup}$ and $\mathsf{alc}_{\sqcap}$ rules. If we consider subset blocking, then the node z is blocked by the root node x. This means that the existential rule $\mathsf{alc}_{\exists}$ is not applied, and hence the expansion stops on this model candidate without generating new individuals. This ABox contains no clash, which means that the tableau procedure will accept A as satisfiable. But this answer is not correct.*

Since the individual z satisfies A, it must also satisfy, due to the GCI in $\mathcal{T}$, the concept $\exists s.\forall r.\forall r.\neg B$; that is, it must have an s successor such that every two-step r successor satisfies $\neg B$. Since r and s are inverses of each other, a two-step s predecessor must satisfy that restriction; hence, every s predecessor of z must do that. See the right side of Figure 2.9, where the dashed arrows represent the r successors obtained by the inverse axiom. This means that y must satisfy $\neg B$, but the ABox contains already $B(y)$, which leads us to a clash in the model candidate. ■

The reason why the procedure was unable to detect the clash was that the node z was not allowed to receive the information that it should satisfy the concept $\forall r.\neg B$, which would be populated upwards by its successor node w through applications of the rule alc_{-}. This *early blocking* problem can be properly solved for this tableau procedure by simply enforcing a stronger condition for blocking, in which the blocked individual must satisfy *exactly* the same concepts as the individual blocking it. More formally, x is *blocked* by y iff $\{D \mid D(x) \in \mathcal{A}\} = \{D \mid D(y) \in \mathcal{A}\}$. This is known as *equality blocking* [HS99].

One should notice that equality blocking can also be applied to the tableau algorithm for satisfiability w.r.t. general TBoxes. Since the condition required for blocking is a stronger one, using it would mean that blocking will come later, and hence one might actually lose in efficiency within an implementation of the method; nonetheless, it would still be sound and complete. It is for this reason that later on, when we formalise the notion of blocking for general tableaux in Chapter 4, we will focus only on equality blocking.

2.4 Automata-Based Decision Algorithms

A different approach for constructing a decision procedure is to use automata to test whether there is a model of the TBox that maps the concept to a non-empty set. Given a logic that has the tree model property, that is, there is a model for an ontology if and only if there is a tree shaped model for the same, the idea is to construct a tree automaton whose accepted language corresponds exactly to those tree-shaped models where the root satisfies the concept being tested. Thus, the language accepted by this automaton is empty if and only if the concept is unsatisfiable.

Before describing how this idea is applied to $\mathcal{ALC}$ w.r.t. $\mathcal{SI}$-TBoxes and LTL, we need to present some basic concepts of automata theory. We are interested in tree automata that work on infinite trees. Intuitively, these automata try to label an input (infinite) tree in such a way that the labeling satisfies the automata acceptance condition (see Definition 2.13). If such a labeling is possible, then the tree is accepted; otherwise it is rejected. Furthermore, when automata are used to decide a property, it is usually sufficient to use unlabeled trees as inputs. This means that, given a fixed arity (i.e., branching factor) k, there is only one such input tree; thus, the language accepted by one of these automata will be either empty or contain the only unlabeled k-ary tree.

Given a positive integer k we use K to denote the set $\{1, \ldots, k\}$. We identify the nodes of the input trees by means of words in K^* in the usual way: the root node is identified by the empty word ε, and the i-th successor of a node u is identified by ui for $1 \leq i \leq k$. The unique unlabeled infinite tree of arity k is represented by the set of all its nodes, namely K^*. As said before, an automaton tries to label the input tree in an appropriate manner. Whenever we are speaking of labeled trees, we will refer to the label of the node $u \in \mathsf{K}^*$ in the tree r by $r(u)$, and in the same fashion we represent an infinite tree r labeled with elements from a set Q as a mapping $r : \mathsf{K}^* \to Q$. We will also use the abbreviation $\overrightarrow{r(u)}$ to denote the tuple $\overrightarrow{r(u)} = (r(u), r(u1), \ldots, r(uk))$. Additionally, we need the concept of a *path* in this tree. A path is a subset $\mathsf{p} \subseteq K^*$ such that $\varepsilon \in \mathsf{p}$ and for every $u \in p$ there is exactly one $i, 1 \leq i \leq k$ with $ui \in \mathsf{p}$.

Definition 2.13 ((Generalised) Büchi tree automaton). *A* generalised Büchi tree automaton *for arity k is a tuple $(Q, \Delta, I, F_1, \ldots, F_n)$, where Q is a finite set of* states*, $\Delta \subseteq Q^{k+1}$ is the* transition relation*, $I \subseteq Q$ is the set of* initial states*, and $F_1, \ldots, F_n \subseteq Q$ are the sets of* final states*. A generalised Büchi tree automaton is called* Büchi automaton *if it has only one set of final states; i.e., if $n = 1$. It is called* looping tree automaton *if $n = 0$.*

A run *of a generalised Büchi automaton on the unlabeled tree K^* is a labeled k-ary tree $r : K^* \to Q$ such that $\overrightarrow{r(u)} \in \Delta$ for all $u \in K^*$. This run is* successful *if for every path p and every $i, 1 \leq i \leq n$, there are infinitely many nodes $u \in \mathsf{p}$ such that $r(u) \in F_i$.* ■

When using automata as decision procedures, one is usually interested in solving the *emptiness problem*, which consists in deciding whether the language accepted by the automaton is empty or not.

Definition 2.14 (Emptiness problem). *The* emptiness problem *for generalised Büchi tree automata for arity k is the problem of deciding whether a given such automaton has a successful run r such that $r(\varepsilon) \in I$ or not.* ■

Although a direct algorithm for deciding the emptiness of a generalised Büchi automaton is sketched in [VW84], no proof of correctness is presented there and in the journal version of that paper [VW86], the idea is simplified by presenting a reduction to the emptiness problem for Büchi automata. In Chapter 5, we will follow a similar approach for computing the so-called behaviour of weighted Büchi automata. First, we will show how to compute the behaviour of weighted Büchi automata. Later, we will introduce a polynomial reduction from weighted generalised Büchi automata to weighted Büchi automata that preserves the behaviour. Our algorithm for computing the behaviour of weighted Büchi automata generalises the well-known ideas employed to decide the emptiness problem in the unweighted case.

The emptiness problem for Büchi automata can be decided in time polynomial in the size of the automaton [Rab70, VW86]. The decision procedure constructs the set of all states that cannot occur as labels in any successful run; we will call these states *bad states*. We can try to disprove that a state is bad by attempting to construct a finite partial run where every path ends in a final state. Every state for which this construction fails is clearly bad, but there may be bad states for which this construction succeeds. The reason is that some of the final states reached by the finite run may themselves bad. Thus, in order to compute all bad states we must iterate this process, where in the next iteration the partial run is required to reach final states that are not already known to be bad. Notice, however, that the construction of a finite partial run ending in non-bad final states can itself be realized by an iterative procedure. Hence, the decision procedure for the emptiness problem uses two nested iterations. In the inner loop, we try to construct a finite partial run finishing in (non-bad) final states for every state. In the outer loop, we use the result of the inner iteration to update the set of (known) bad states, and then re-start the inner iteration with this new information.

Let us call the states for which there is a finite partial run finishing in non-bad final states *adequate*. First, any state $q \in Q$ for which there is a transition leading to only non-bad final states is clearly adequate. Then, every state for which there is a transition leading only to states that are either (i) final and not bad or (ii) already known to be adequate is also adequate. Obviously, during this iteration, the set of adequate states becomes stable after at most $|Q|$ iterations. The outer loop then adds all the states that were found not to be adequate to the set of bad states. The set of bad states maintained in this outer iteration becomes stable after at most $|Q|$ steps. This yields an emptiness test that runs in time polynomial in the number of states (see [VW86] for details). In the case of looping automata, this method can be simplified to a single bottom-up iteration [BT01].

In the following subsections, we will show how we can use automata, and in particular the emptiness test just sketched, to decide satisfiability of $\mathcal{ALC}$ concept terms w.r.t. $\mathcal{SI}$-TBoxes, as well as axiomatic satisfiability of LTL formulae.

2.4.1 Satisfiability of $\mathcal{ALC}$ Concepts with $\mathcal{SI}$-TBoxes

The automata-based approach for deciding satisfiability of an $\mathcal{ALC}$ concept term w.r.t. a general $\mathcal{SI}$-TBox is based on the fact that a concept is satisfiable iff it has a so-called Hintikka tree, which is basically a tree model where every node is labeled with the concept terms to which it belongs. Given a concept C and an $\mathcal{SI}$-TBox, we will construct a looping tree automaton whose successful runs correspond exactly to the Hintikka trees.

In order to simplify the notation, we assume that every concept term is presented in *negation normal form* (NNF); that is, negation appears only in front of concept names. This assumption has no impact in the generality of the method as every $\mathcal{ALC}$ concept term can be transformed into NNF in linear time using the de Morgan laws, duality of quantifiers and elimination of double negations. We will denote the NNF of a concept term C as $\mathsf{nnf}(C)$ and $\mathsf{nnf}(\neg C)$ as $\backsim C$. Given an $\mathcal{ALC}$ concept term C and a general $\mathcal{SI}$-TBox $\mathcal{T}$, we will use the abbreviation $\mathsf{sub}(C, \mathcal{T})$ to denote the set containing all the subconcepts of C as well as of the concept $\backsim D \sqcup E$ for $D \sqsubseteq E \in \mathcal{T}$.

The automaton we construct for deciding satisfiability of concepts w.r.t. general $\mathcal{SI}$-TBoxes will have so-called *Hintikka sets* as states. Hintikka sets contain as elements subconcepts of the input concept and TBox, as well as information about the transitivity of certain roles. For this, we will additionally use $\mathsf{rol}(C, \mathcal{T})$ to denote the set of all role names appearing in C or in $\mathcal{T}$.

Definition 2.15 ($\mathcal{SI}$-Hintikka set). *A set $H \subseteq \mathsf{sub}(C, \mathcal{T}) \cup \mathsf{rol}(C, \mathcal{T})$ is called an* $\mathcal{SI}$-Hintikka set *for $(C, \mathcal{T})$ if the following three conditions are satisfied:*

(i) if $D \sqcap E \in H$, then $\{D, E\} \subseteq H$;

(ii) if $D \sqcup E \in H$, then $\{D, E\} \cap H \neq \emptyset$; and

(iii) there is no concept name $A \in \mathsf{CN}$ such that $\{A, \neg A\} \subseteq H$.

An $\mathcal{SI}$-Hintikka set H is compatible with the GCI $D \sqsubseteq E \in \mathcal{T}$ *iff either $H = \emptyset$ or $\backsim D \sqcup E \in H$. It is* compatible with the transitivity axiom $\mathsf{trans}(r) \in \mathcal{T}$ iff $H = \emptyset$ *or $r \in H$. Finally, H is* compatible with the inverse axiom $\mathsf{inv}(r, s) \in \mathcal{T}$ *iff it holds that $r \in H$ if and only if $s \in H$.* ■

The arity k of the input accepted by our automaton is given by the number of existential restrictions, i.e., concept terms of the form $\exists r.D$, present in $\mathsf{sub}(C, \mathcal{T})$. For the transition relation, it will be important to know which successor in the tree corresponds to which existential restriction being satisfied; for that reason, we fix an arbitrary bijection $\varphi : \{\exists r.D \mid \exists r.D \in \mathsf{sub}(C, \mathcal{T})\} \to \mathsf{K}$. A Hintikka tree is a k-ary tree labeled with Hintikka sets that satisfies additional compatibility conditions dealing with the existential- and value restrictions appearing in its node labels. To obtain full k-ary trees, we will add dummy nodes labeled with the empty set (which is itself an $\mathcal{SI}$-Hintikka set, and compatible with every axiom) where appropriate.

Definition 2.16 (Hintikka condition). *The tuple $(H_0, H_1, \ldots, H_k)$ of Hintikka sets for $(C, \mathcal{T})$ satisfies the* Hintikka condition *iff the following two conditions hold for every existential restriction $\exists r.D \in \mathsf{sub}(C, \mathcal{T})$:*

- *if* $\exists r.D \in H_0$, *then* $H_{\varphi(\exists r.D)}$ *contains* D *as well as every* E *for which there is a value restriction* $\forall r.E \in H_0$; *if, additionally,* $r \in H_0$, *then also* $\forall r.E$ *belongs to* $H_{\varphi(\exists r.D)}$ *for all value restriction* $\forall r.E \in H_0$; *and*
- *if* $\exists r.D \notin H_0$, *then* $H_{\varphi(\exists r.D)} = \emptyset$.

A tuple satisfying the $\mathcal{SI}$*-Hintikka condition is called* compatible with the GCI $D \sqsubseteq E \in \mathcal{T}$ *(respectively* compatible with the transitivity axiom $\mathsf{trans}(r) \in \mathcal{T}$*) if all its components are compatible with* $D \sqsubseteq E$ *(compatible with* $\mathsf{trans}(r)$*, respectively). It is* compatible with the inverse axiom $\mathsf{inv}(r, r') \in \mathcal{T}$ *if all its components are compatible with* $\mathsf{inv}(r, r')$ *and the following holds for all* $s \in \{r, r'\}$ *and* $s^- \in \{r, r'\} \setminus \{s\}$*: for every* $\forall s.F \in H_{\varphi(\exists s^-.D)}$*, the set* H_0 *contains* F *and additionally* $\forall s.F$ *if* $s \in H_0$.

A tuple of $\mathcal{SI}$*-Hintikka sets that satisfies the* $\mathcal{SI}$*-Hintikka condition is* compatible *with a general* $\mathcal{SI}$*-TBox* $\mathcal{T}$ *if it is compatible with every axiom* $t \in \mathcal{T}$. ■

We can now formally define Hintikka trees.

Definition 2.17 (Hintikka tree). *A* Hintikka tree *for* $(C, \mathcal{T})$ *is a* k*-ary tree* $\mathcal{H}$ *labeled with Hintikka sets for* $(C, \mathcal{T})$ *such that* $C \in \mathcal{H}(\varepsilon)$ *and for every node* $u \in \mathsf{K}^*$ *the tuple* $\overrightarrow{\mathcal{H}(u)}$ *is compatible with* $\mathcal{T}$. ■

The following result shows that testing for satisfiability of a concept C w.r.t. an $\mathcal{SI}$-TBox $\mathcal{T}$ is equivalent to deciding the existence of an $\mathcal{SI}$-Hintikka tree for $(C, \mathcal{T})$. This lemma can be shown by a simple adaptation of the arguments presented previously in [BHP07, BHP08].

Lemma 2.18. *A concept* C *is satisfiable w.r.t. a general* $\mathcal{SI}$*-TBox* $\mathcal{T}$ *iff there is a Hintikka tree for* $(C, \mathcal{T})$.

Given this lemma, we now know that it is enough to construct an automaton whose successful runs correspond to such Hintikka trees. We can then test for satisfiability of the concept w.r.t. a $\mathcal{SI}$-TBox by performing an emptiness test on this automaton. In this case, a looping automaton suffices for deciding the property.

Definition 2.19 (Automaton $\mathcal{A}^{\mathsf{sat}}_{C,\mathcal{T}}$). *Let* C *be an* $\mathcal{ALC}$ *concept term,* $\mathcal{T}$ *a general* $\mathcal{SI}$*-TBox and* k *the number of existential restrictions in* $\mathsf{sub}(C, \mathcal{T})$. *The looping automaton* $\mathcal{A}^{\mathsf{sat}}_{C,\mathcal{T}}$ *is given by* $\mathcal{A}^{\mathsf{sat}}_{C,\mathcal{T}} = (Q, \Delta, I)$ *where*

- Q *is the set of all Hintikka sets for* $(C, \mathcal{T})$;
- Δ *is the set of all tuples* $(H_0, H_1, \ldots, H_k) \in Q^{k+1}$ *that satisfy the Hintikka condition and are compatible with* $\mathcal{T}$; *and*
- $I = \{H \in Q \mid C \in H\}$.

■

As expected, the successful runs of this automaton where the root is labeled with an element of I correspond exactly to $\mathcal{SI}$-Hintikka trees for $(C, \mathcal{T})$. This yields the following result [BHP08].

Theorem 2.20. *Let C be an $\mathcal{ALC}$ concept term and $\mathcal{T}$ an $\mathcal{SI}$-TBox. The automaton $\mathcal{A}^{\mathsf{sat}}_{C,\mathcal{T}}$ has a successful run r with $r(\varepsilon) \in I$ iff C is satisfiable w.r.t. $\mathcal{T}$.*

This theorem shows that the emptiness test sketched before can be used as a decision procedure for satisfiability of $\mathcal{ALC}$ concept terms w.r.t. $\mathcal{SI}$-TBoxes. The automation $\mathcal{A}_{C,\mathcal{T}}$ is a looping automaton, that is, it makes no use of the Büchi acceptance condition on runs. The automata construction we will show in the next subsection for deciding axiomatic satisfiability of LTL formulae requires these acceptance conditions for correctness.

2.4.2 Axiomatic Satisfiability of LTL Formulae

In order to decide axiomatic satisfiability of LTL formulae, we will construct an automaton whose successful runs correspond to computations for the input. Notice that a computation $\pi : \mathbb{N} \to \mathscr{P}(\mathsf{P})$ can be seen also as a unary tree, that is, a tree where every node has exactly one successor. More precisely, each node represents one point in time and the successor relation in this tree is given by the standard ordering of natural numbers. Thus, the automaton we construct will have the unique unlabeled unary tree as input. The states of this automaton will be sets of LTL formulae, which intuitively represent the set of all formulae that are satisfied at a given point in time. In that sense, these states correspond to the Hintikka sets defined in the previous subsection. Notice nonetheless that this correspondence will not be precise since for LTL we will follow the ideas of previous automata constructions (e.g. [WVS83]), and hence will not assume that the formulae are in negation normal form. Given an LTL formula ϕ and a set of LTL formulae $\mathcal{R}$, we define the *closure of* $(\phi, \mathcal{R})$ as the set of all subformulae of ϕ and $\mathcal{R}$, and their negations, where double negations are cancelled. This set is denoted by $\mathsf{cl}(\phi, \mathcal{R})$.

The states of our automaton are so-called *elementary* sets of formulae, which play the role of the Hintikka sets of the previous subsection; that is, they are maximal and consistent sets of subformulae in $\mathsf{cl}(\phi, \mathcal{R})$.

Definition 2.21 (Elementary set). *A set $H \subseteq \mathsf{cl}(\phi, \mathcal{R})$ is called an* elementary set *for $(\phi, \mathcal{R})$ if it satisfies the following conditions:*

- $\neg\phi \in H$ *iff* $\phi \notin H$;
- $\phi \wedge \psi \in H$ *iff* $\{\phi, \psi\} \subseteq H$;
- $\psi \in H$ *implies* $\phi \mathcal{U} \psi \in H$;
- *if* $\phi \mathcal{U} \psi \in H$ *and* $\psi \notin H$, *then* $\phi \in H$

■

As we have said before, the automaton for satisfiability of LTL formulae will take unary trees as inputs; i.e., its runs will be infinite *words* over the set of states. The transition relation is thus binary. This transition relation makes sure that the temporal operators are adequately propagated to the successor nodes; for instance, if we have a

next formula $\bigcirc\psi$ in the label of a node, then its successor node must contain ψ. This is formalised by the following definition.

Definition 2.22 (Compatible). *A tuple (H, H') of elementary sets is called* compatible *iff it satisfies the following conditions:*

- *for all $\bigcirc\psi \in \mathsf{cl}(\phi, \mathcal{R})$, $\bigcirc\psi \in H$ iff $\psi \in H'$; and*
- *for all $\psi_1 \mathcal{U} \psi_2 \in \mathsf{cl}(\phi, \mathcal{R})$, $\psi_1 \mathcal{U} \psi_2 \in H$ iff either (i) $\psi_2 \in H$ or (ii) $\psi_1 \in H$ and $\psi_1 \mathcal{U} \psi_2 \in H'$.*

■

The runs of our automaton will be sequences of elementary sets where each two consecutive ones form a compatible tuple. In contrast to the case for $\mathcal{SI}$, the presence of a run of this automaton does not imply the existence of a computation. The reason is that one can delay the satisfaction of an *until* formula indefinitely; that is, every node in the run may have the formula $\psi_1 \mathcal{U} \psi_2$ while none has ψ_2, violating this way the last condition in the definition of a computation for the input (see Definition 2.9). In order to rule out these kinds of runs and make sure that each *until* formula is eventually satisfied, we will impose a generalised Büchi condition which introduces a set of final states for each until formula in $\mathsf{cl}(\phi, \mathcal{R})$. Intuitively, each such set of final states is in charge of enforcing the eventual satisfaction of one specific until formula.

Definition 2.23 (Automaton $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}}$). *Let ϕ and $\mathcal{R}$ be an LTL formula and a set of LTL formulae, respectively, and let $\theta_1 \mathcal{U} \psi_1, \ldots, \theta_n \mathcal{U} \psi_n$ be all the until formulae in $\mathsf{cl}(\phi, \mathcal{R})$. The generalised Büchi automaton $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}} := (Q, \Delta, I, F_1, \ldots, F_n)$ is given by*

- *Q is the set of all elementary sets for $(\phi, \mathcal{R})$;*
- *Δ consists of all compatible pairs $(H, H') \in Q \times Q$;*
- *$I := \{H \in Q \mid \mathcal{R} \cup \{\phi\} \subseteq H\}$;*
- *for $1 \leq i \leq n$, $F_i := \{H \in Q \mid \psi_i \in H \text{ or } \theta_i \mathcal{U} \psi_i \notin H\}$.*

■

The successful runs of this automaton whose root is labelled with an initial state correspond to the computations for the input $(\phi, \mathcal{R})$. From this, we obtain the following result [WVS83].

Theorem 2.24. *Let ϕ be an LTL formula and $\mathcal{R}$ a set of LTL formulae. The automaton $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}}$ has a successful run r with $r(\varepsilon) \in I$ iff ϕ is axiomatic satisfiable w.r.t. $\mathcal{R}$.*

From this theorem it follows that axiomatic satisfiability of LTL formulae can be decided by an emptiness test on the automaton $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}}$.

In this chapter we have described several previously known algorithms for reasoning in different logics, starting from the fairly inexpressive $\mathcal{HL}$ all the way up to the

inclusion of more complex constructors and axioms restricting the interpretations for concepts and roles in DLs. We then left the DL family to include also the temporal operators for LTL.

Broadly, we showed the main characteristics of two different approaches for constructing decision procedures. On one hand, the tableau-based method, that tries to construct a model while keeping the restrictions imposed by the axioms (included as expansion rules). On the other hand is the automata-based approach that tries to construct an automaton for which an emptiness test leads to a correct decision.

The particular instances of decision procedures presented in this chapter will help us formalise the notions of general tableau algorithms (in Chapter 3) and so-called axiomatic automata (in Chapter 5), respectively. We will then show how each of these decision procedures can be modified to obtain what is called a pinpointing procedure; intuitively, one that will allow us to deduce how the presence of certain axioms influences the property being tested. The output of a pinpointing procedure will be the so-called pinpointing formula, from which all explanations and diagnoses can be inferred.

Chapter 3

Tableaux and Pinpointing

The previous chapter introduced procedures that allow us to decide if a property, such as subsumption or satisfiability of concept names, follows from a set of axioms. The sets of axioms used could take very different shapes; namely, concept definitions, assertional axioms, or GCIs, in the case of DLs, or LTL formulae. The decision procedures we presented came in two flavours: the tableau-like and the automata-based procedures. It is the goal of this work to show how to extend them in such a way that, once a decision is made, we are able to justify it by retrieving those axioms that are relevant for the obtained answer. The approach followed in this work consists on finding a monotone Boolean formula, which we call *pinpointing formula*, from which the desired sets of axioms can be deduced. The present and following chapters will deal with the tableau-like methods, while we delay the treatment of automata-based procedures until Chapter 5.

Before we can begin with the task of extending any kind of algorithm, we need to formally describe the problem that we are trying to solve; namely, the properties that should be satisfied by the pinpointing formula. This in turn will require a formal definition of the kinds of properties that the original procedures decide. All these notions are introduced in Section 3.1.

Afterwards, we proceed to describe extensions of tableau-like decision procedures that compute the desired pinpointing formula. In order to improve understanding, this is done in two steps. We first focus in the special case of *ground tableaux* of which the subsumption algorithm of Section 2.3.1 is an instance. We then generalise all the notions and results to what we call *general tableaux* in Section 3.3. This notion encompasses the procedures described in Sections 2.3.2 and 2.3.3, but is not able to deal with blocking conditions as described in the last two sections of the previous chapter. The pinpointing extensions of general tableaux are shown to correctly compute a pinpointing formula whenever they terminate.

The extension presented in this chapter follows the ideas introduced by Baader and Hollunder in [BH95]. There, the consistency algorithm for $\mathcal{ALC}$ ABoxes is extended by a labelling technique that ultimately computes a pinpointing formula. A similar approach was followed by Schlobach and Cornet [SC03] for concept unsatisfiability with respect to so-called unfoldable $\mathcal{ALC}$ terminologies. The main difference between

Baader and Hollunder's approach and that by Schlobach and Cornet is that the latter tries to find the sets of axioms that are relevant to unsatisfiability directly, rather than by using the intermediary pinpointing formula as done in the former approach. In reality, the result obtained using the method in [SC03] can be seen as a pinpointing formula written in disjunctive normal form. Although these ideas have been extended to include additional constructors or use different kinds of axioms (see, for instance, [PSK05, MLBP06]), each of these extensions has been made to work specifically for the language being studied. Nonetheless, except for the case dealing with blocking [LMP06] that needs special attention, they all follow the same basic ideas.

Unfortunately, as shown at the end of this chapter, there is no warranty that the extended algorithm will stop after a finite number of steps, even if the original tableau does. This fact is specially relevant since none of the papers cited so far deals with termination of the extensions they present. Actually, termination is usually disregarded as trivially following from the same causes of termination of the original tableau, giving no further insight into which these causes are in reality. It will be the task of Chapter 4 to introduce a framework where both, tableaux and their pinpointing extensions, are guaranteed to terminate. It is in that chapter too that we will introduce the notion of blocking for general tableaux and their pinpointing extensions.

3.1 Basic Notions for Pinpointing

We begin this section by defining the general form of the inputs for the decision algorithms used along this work. These inputs, called *axiomatised inputs*, consist of two parts. Intuitively, one part corresponds to a knowledge base, that is, a set of axioms possibly restricted to satisfy additional internal restrictions, and the other expresses the instance of the inference problem that needs to be tested against this knowledge base. The internal restrictions in the set of axioms are necessary for modelling e.g. acyclic- or $\mathcal{SI}$-TBoxes, where not every set of axioms is allowed. Indeed, acyclic TBoxes require every concept name to appear at most one in the left-hand-side of a concept definition, and $\mathcal{SI}$-TBoxes are restricted to allow the use of each role name in at most one inverse axioms. But notice that in both cases, if a set of axioms is allowed to be used as a knowledge base, then any of its subsets is also allowed. In our general approach we keep this property.

The consequences in which we are interested need to satisfy a monotonicity restriction in the sense that adding axioms to the knowledge base can only make more consequences true, but not falsify any that already follows from the original set of axioms. A property is merely a set of axiomatised inputs, and the decision problem associated with such property consist on deciding, for a given axiomatised input, whether it belongs to the set or not. A property that models consequences satisfying the monotonicity restriction stated above will be called *consequence property*.

Definition 3.1 (Axiomatised input, c-property). *Let $\mathfrak{I}$ be a set, called the set of* inputs*, $\mathfrak{T}$ be a set, called the set of* axioms*, and let $\mathscr{P}_{admis}(\mathfrak{T}) \subseteq \mathscr{P}_{fin}(\mathfrak{T})$ be a set of finite subsets of $\mathfrak{T}$. $\mathscr{P}_{admis}(\mathfrak{T})$ is called* admissible *if $\mathcal{T} \in \mathscr{P}_{admis}(\mathfrak{T})$ implies $\mathcal{T}' \in \mathscr{P}_{admis}(\mathfrak{T})$ for all $\mathcal{T}' \subseteq \mathcal{T}$. An* axiomatised input *for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$ is of the*

form $(\mathcal{I}, \mathcal{T})$ *where* $\mathcal{I} \in \mathfrak{I}$ *and* $\mathcal{T} \in \mathscr{P}_{admis}(\mathfrak{T})$.

A consequence property *(or* c-property *for short) is a set* $\mathcal{P} \subseteq \mathfrak{I} \times \mathscr{P}_{admis}(\mathfrak{T})$ *such that* $(\mathcal{I}, \mathcal{T}) \in \mathcal{P}$ *implies* $(\mathcal{I}, \mathcal{T}') \in \mathcal{P}$ *for every* $\mathcal{T}' \in \mathscr{P}_{admis}(\mathfrak{T})$ *with* $\mathcal{T}' \supseteq \mathcal{T}$. ■

The idea behind c-properties on axiomatised inputs is to model consequence relations in logic, i.e., the c-property $\mathcal{P}$ holds if the input $\mathcal{I}$ "follows" from the axioms in $\mathcal{T}$. The monotonicity requirement on c-properties corresponds to the fact that we want to restrict the attention to consequence relations induced by monotonic logics. In fact, for non-monotonic logics, looking at minimal sets of axioms that have a given consequence does not make much sense.

To illustrate Definition 3.1, consider the set $\mathsf{N_C}$ of concept names. Assume that $\mathfrak{I}$ is the set of ordered pairs $\mathsf{N_C} \times \mathsf{N_C}$ and that $\mathfrak{T}$ consists of all $\mathcal{HL}$-GCIs over these concept names. Then the following is a c-property according to the above definition: $\mathcal{P} := \{((C, D), \mathcal{T}) \mid C \sqsubseteq_{\mathcal{T}} D\}$. This property represents subsumption w.r.t. general $\mathcal{HL}$-TBoxes. As a concrete example, consider $\Gamma := ((A, B), \mathcal{T})$ where $\mathcal{T}$ consists of the following GCIs:

$$\text{ax}_1\colon\ A \sqsubseteq C, \qquad \text{ax}_2\colon\ A \sqsubseteq D, \qquad \text{ax}_3\colon\ D \sqsubseteq C, \qquad \text{ax}_4\colon\ C \sqcap D \sqsubseteq B \tag{3.1}$$

It is easy to see that $\Gamma \in \mathcal{P}$. Note that Definition 3.1 is general enough to capture other variants of the example above, for instance, where $\mathfrak{I}'$ consist of tuples of the form $(C, D, \mathcal{T}_1) \in \mathcal{I} \times \mathscr{P}_{fin}(\mathfrak{T})$ and the c-property is defined as

$$\mathcal{P}' := \{((C, D, \mathcal{T}_1), \mathcal{T}_2) \mid C \sqsubseteq_{\mathcal{T}_1 \cup \mathcal{T}_2} D\}.$$

For example, if we take the axiomatised input $\Gamma' := ((A, B, \{\text{ax}_3, \text{ax}_4\}), \{\text{ax}_1, \text{ax}_2\})$, then $\Gamma' \in \mathcal{P}'$.

Due to the monotonicity of c-properties, it may well be that some axioms are irrelevant for deducing a consequence. If we are interested in justifying such a consequence, we would need to get rid of all those irrelevant axioms and present a minimal knowledge base from which the consequence still follows. If, on the contrary, the consequence is detected as an error, we might want to remove only enough axioms to get rid of it but not more, since that might also remove some desired consequences.

Definition 3.2 (MinA,MaNA). *Given an axiomatised input* $\Gamma = (\mathcal{I}, \mathcal{T})$ *and a c-property* $\mathcal{P}$*, a set of axioms* $\mathcal{S} \subseteq \mathcal{T}$ *is called a* minimal axiom set (MinA) for Γ w.r.t. $\mathcal{P}$ *if* $(\mathcal{I}, \mathcal{S}) \in \mathcal{P}$ *and* $(\mathcal{I}, \mathcal{S}') \notin \mathcal{P}$ *for every* $\mathcal{S}' \subset \mathcal{S}$*. Dually, a set of axioms* $\mathcal{S} \subseteq \mathcal{T}$ *is called a* maximal non-axiom set (MaNA) for Γ w.r.t. $\mathcal{P}$ *if* $(\mathcal{I}, \mathcal{S}) \notin \mathcal{P}$ *and* $(\mathcal{I}, \mathcal{S}') \in \mathcal{P}$ *for every* $\mathcal{T} \supseteq \mathcal{S}' \supset \mathcal{S}$*. The set of all MinAs (MaNAs) for* Γ *w.r.t.* $\mathcal{P}$ *will be denoted as* $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ *(*$\mathsf{MAX}_{\mathcal{P}(\Gamma)}$*).* ■

Note that the notions of MinA and MaNA are only interesting in the case where $\Gamma \in \mathcal{P}$. In fact, otherwise the monotonicity property satisfied by $\mathcal{P}$ implies that $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \emptyset$ and $\mathsf{MAX}_{\mathcal{P}(\Gamma)} = \{\mathcal{T}\}$. In the above example, where we have $\Gamma \in \mathcal{P}$, it is easy to see that $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \{\{\text{ax}_1, \text{ax}_2, \text{ax}_4\},\ \{\text{ax}_2, \text{ax}_3, \text{ax}_4\}\}$. In the variant of the example where only subsets of $\{\text{ax}_1, \text{ax}_2\}$ can be taken, we have $\mathsf{MIN}_{\mathcal{P}'(\Gamma')} = \{\{\text{ax}_2\}\}$.

The set $\mathsf{MAX}_{\mathcal{P}(\Gamma)}$ can be obtained from $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ by computing the minimal hitting sets of $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$, and then complementing these sets [SC03, LS05]. A set $\mathcal{S} \subseteq \mathcal{T}$ is a *hitting set* of $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ if it has a nonempty intersection with every element of $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$, and is a *minimal hitting set* if no strict subset of $\mathcal{S}$ is itself a hitting set. In our example, the minimal hitting sets of $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ are $\{\mathrm{ax}_1, \mathrm{ax}_3\}, \{\mathrm{ax}_2\}, \{\mathrm{ax}_4\}$, and thus $\mathsf{MAX}_{\mathcal{P}(\Gamma)} = \{\{\mathrm{ax}_2, \mathrm{ax}_4\},\ \{\mathrm{ax}_1, \mathrm{ax}_3, \mathrm{ax}_4\},\ \{\mathrm{ax}_1, \mathrm{ax}_2, \mathrm{ax}_3\}\}$. The intuition behind this reductions is that, to get a set of axioms that does not have the consequence, we must remove from $\mathcal{T}$ at least one axiom for every MinA, and thus the minimal hitting sets give us the minimal sets to be removed.

The reduction we have just sketched shows that it is enough to design an algorithm for computing all MinAs, since the MaNAs can then be obtained by a hitting set computation. It should be noted, however, that this reduction is not polynomial: there may be exponentially many hitting sets of a given collection of sets, and even deciding whether such a collection has a hitting set of cardinality $\leq n$ is already an NP-complete problem [GJ79]. Also note that there is a similar reduction involving hitting sets for computing the MinAs from all MaNAs.

Instead of computing MinAs or MaNAs, one can also compute the pinpointing formula.[10] To define the pinpointing formula, we assume that every axiom $t \in \mathcal{T}$ is labeled with a unique propositional variable, which we denote as $\mathsf{lab}(t)$. Let $\mathsf{lab}(\mathcal{T})$ be the set of all propositional variables labeling an axiom in $\mathcal{T}$. A *monotone Boolean formula* over $\mathsf{lab}(\mathcal{T})$ is a Boolean formula using (some of) the variables in $\mathsf{lab}(\mathcal{T})$ and only the connectives conjunction and disjunction. We further assume that the formula $\top$, which is always evaluated as true, is a monotone Boolean formula. As usual, we identify a propositional *valuation* with the set of propositional variables it makes true. For a valuation $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$, let $\mathcal{T}_\mathcal{V} := \{t \in \mathcal{T} \mid \mathsf{lab}(t) \in \mathcal{V}\}$.

Definition 3.3 (Pinpointing formula). *Given a c-property $\mathcal{P}$ and an axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$, a monotone Boolean formula ϕ over $\mathsf{lab}(\mathcal{T})$ is called a* pinpointing formula *for $\mathcal{P}$ and Γ if the following holds for every valuation $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$: $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$ iff $\mathcal{V}$ satisfies ϕ.* ■

In our example, we can take $\mathsf{lab}(\mathcal{T}) = \{\mathrm{ax}_1, \ldots, \mathrm{ax}_4\}$ as the set of propositional variables. It is easy to see that $(\mathrm{ax}_1 \vee \mathrm{ax}_3) \wedge \mathrm{ax}_2 \wedge \mathrm{ax}_4$ is a pinpointing formula for $\mathcal{P}$ and Γ.

Valuations have a natural partial order by means of set inclusion, which allows us to speak about *minimal* and *maximal* valuations. The following is an immediate consequence of the definition of a pinpointing formula [BH95].

Lemma 3.4. *Let $\mathcal{P}$ be a c-property, $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatised input, and ϕ a pinpointing formula for $\mathcal{P}$ and Γ. Then*

$$\begin{aligned} \mathsf{MIN}_{\mathcal{P}(\Gamma)} &= \{\mathcal{T}_\mathcal{V} \mid \mathcal{V} \textit{ is a minimal valuation satisfying } \phi\} \\ \mathsf{MAX}_{\mathcal{P}(\Gamma)} &= \{\mathcal{T}_\mathcal{V} \mid \mathcal{V} \textit{ is a maximal valuation falsifying } \phi\} \end{aligned}$$

[10] This corresponds to what was called the *clash formula* in [BH95]. Here, we distinguish between the pinpointing formula, which can be defined independently of a tableau algorithm, and the clash formula, which is induced by a run of a specific tableau algorithm.

This lemma shows that it is enough to design an algorithm for computing a pinpointing formula to obtain all MinAs and MaNAs. However, like the previous reduction for computing $\mathsf{MAX}_{\mathcal{P}(\Gamma)}$ from $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$, the reduction suggested by the lemma is not polynomial. For example, to obtain $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ from ϕ, one can bring ϕ into disjunctive normal form and then remove disjuncts implying other disjuncts. It is well-known that this can cause an exponential blowup. Conversely, however, the set $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ can directly be translated into the pinpointing formula

$$\bigvee_{\mathcal{S} \in \mathsf{MIN}_{\mathcal{P}(\Gamma)}} \bigwedge_{s \in \mathcal{S}} \mathsf{lab}(s). \tag{3.2}$$

Returning to our example, the pinpointing formula obtained in this fashion from $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \{\{\mathrm{ax}_1, \mathrm{ax}_2, \mathrm{ax}_4\}, \{\mathrm{ax}_2, \mathrm{ax}_3, \mathrm{ax}_4\}\}$ is $(\mathrm{ax}_1 \wedge \mathrm{ax}_2 \wedge \mathrm{ax}_4) \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3 \wedge \mathrm{ax}_4)$, which is equivalent to the pinpointing formula we had directly computed.

3.2 Pinpointing in Ground Tableaux

Before describing how general tableau-based algorithms can be extended to procedures that compute a pinpointing formula, we show how this is done in a restricted case that we will call *ground tableaux*. This case is still interesting by itself, since it encompasses several decision procedures, such as the subsumption algorithm for $\mathcal{HL}$ or the congruence closure algorithm [NO07]. The proofs of all the results presented in this section will be delayed to the more general statements of Section 3.3.

Definition 3.5 (Ground tableau). *Let $\mathfrak{I}$ be a set of inputs and $\mathscr{P}_{admis}(\mathfrak{T})$ an admissible set of sets of elements in $\mathfrak{T}$. A* ground tableau *for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$ is a tuple $S = (\Sigma, \cdot^S, \mathcal{R}, \mathcal{C})$ where*

- *Σ is a set called a* signature*;*
- *$\cdot^S$ is a function, called the* initial function*, that maps every $\mathcal{I} \in \mathfrak{I}$ and every $t \in \mathfrak{T}$ to a finite subset of Σ;*
- *$\mathcal{R}$ is a set of* rules *of the form $(B_0, \mathcal{S}) \to B$ where B_0 and B are finite subsets of Σ and $\mathcal{S}$ is a finite set of axioms;*
- *$\mathcal{C}$ is a set of finite subsets of Σ, called* clashes*.*

■

A ground tableau decides a property with the help of so-called S-states that intuitively contain all the knowledge that has been deduced during the execution of the method. An *S-state* is a pair $\mathfrak{S} = (A, \mathcal{T})$ where A is a finite subset of Σ and $\mathcal{T} \in \mathscr{P}_{admis}(\mathfrak{T})$ is an admissible set of axioms. In this case, we call A and $\mathcal{T}$ the *assertion-* and *axiom*-component of $\mathfrak{S}$, respectively. The elements of A are also called *assertions*. The decision procedure begins with the *initial state* $(\mathcal{I}, \mathcal{T})^S$ that depends

on the axiomatised input $(\mathcal{I}, \mathcal{T})$ given to the algorithm. This state is found extending the initial function $\cdot^S$ as follows:

$$(\mathcal{I}, \mathcal{T})^S = (\mathcal{I}^S \cup \bigcup_{t \in \mathcal{T}} t^S, \mathcal{T}).$$

Consider for example the procedure for deciding subsumption of $\mathcal{HL}$ concepts described in Section 2.3.1. This algorithm stores all the information needed to make the decision in a set of pairs of the form $(\mathbf{A}, \mathbf{B})$, where $\mathbf{A}, \mathbf{B}$ are concept names. We can thus consider its signature to be formed by all such pairs. That algorithm begins with all the *trivial* knowledge stating that every concept appearing in the input set of axioms is subsumed by itself. We can do this by fixing the initial function to map every axiom t of the form $\prod_{i=1}^{n} \mathbf{A}_i \sqsubseteq \prod_{j=1}^{m} \mathbf{B}_j$ to the set

$$t^S = \{(\mathbf{A}_i, \mathbf{A}_i) \mid 1 \leq i \leq n\} \cup \{(\mathbf{B}_j, \mathbf{B}_j) \mid 1 \leq j \leq m\}.$$

Now, since we want this procedure to work for every subsumption relation we desire to test, and the decision made by such ground tableaux relies only on the information stored in its states, we need a way to specify which specific subsumption relation is the one we are currently trying to decide. For this reason, we extend the signature to also include assertions of the form $\mathbf{A} \sqsubseteq^? \mathbf{B}$ with $\mathbf{A}, \mathbf{B}$ concept names. The presence of an assertion of this kind specifies the request for deciding the subsumption of $\mathbf{A}$ by $\mathbf{B}$. If we consider the encoding of these inputs as presented in Page 35, then the initial function must map every input of the form $(\mathbf{A}, \mathbf{B})$ asking for a subsumption test to the set containing the corresponding assertion $\mathbf{A} \sqsubseteq^? \mathbf{B}$. More precisely, if we take the axiomatised input $\Gamma = ((\mathbf{A}, \mathbf{B}), \mathcal{T})$, where $\mathcal{T}$ contains the axioms in (3.1), then the initial function produces the S-state

$$\Gamma^S = (\{\mathbf{A} \sqsubseteq^? \mathbf{B}, (\mathbf{A}, \mathbf{A}), (\mathbf{B}, \mathbf{B}), (\mathbf{C}, \mathbf{C}), (\mathbf{D}, \mathbf{D})\}, \mathcal{T}).$$

The rules in $\mathcal{R}$ are used then to iteratively extend the first component of an S-state $\mathfrak{S}$ depending exclusively on the assertions and axioms appearing in $\mathfrak{S}$. Returning to the subsumption procedure, the rule hl specifies, intuitively, that whenever we know that a concept name $\mathbf{A}$ is subsumed by all the $\mathbf{A}_i$s, and the conjunction of those $\mathbf{A}_i$s is subsumed by the conjunction of some $\mathbf{B}_j$s by means of an axiom in $\mathcal{T}$, then we can deduce that $\mathbf{A}$ is also subsumed by each of the $\mathbf{B}_j$, and we can thus extend our explicit knowledge accordingly. More concretely, since the S-state Γ^S described above contains the assertion $(\mathbf{A}, \mathbf{A})$ and the axiom $\mathbf{A} \sqsubseteq \mathbf{D}$, a rule application would add the assertion $(\mathbf{A}, \mathbf{D})$ to it. That rule can be rewritten in a tableau-like shape as follows:

$$\mathsf{hl} : (\{(\mathbf{A}, \mathbf{A}_i) \mid 1 \leq i \leq n\}, \{\prod_{i=1}^{n} \mathbf{A}_i \sqsubseteq \prod_{j=1}^{m} \mathbf{B}_j\}) \to \{(\mathbf{A}, \mathbf{B}_j) \mid 1 \leq j \leq m\}.$$

The following definition formalises this behaviour.

Definition 3.6 (Rule application). *Given an S-state $\mathfrak{S} = (A, \mathcal{T})$, and a rule $\mathsf{R} : (B_0, \mathcal{S}) \to B$ we say that R is* applicable to $\mathfrak{S}$ *if the following three conditions are satisfied: (i) $\mathcal{S} \subseteq \mathcal{T}$, (ii) $B_0 \subseteq A$, and (iii) $B \not\subseteq A$.*

If the rule R is applicable to the S-state $\mathfrak{S} = (A, \mathcal{T})$, then the application of R to $\mathfrak{S}$ *yields the new S-state $(A \cup B, \mathcal{T})$. If $\mathfrak{S}'$ is obtained from $\mathfrak{S}$ by the application of the rule R, then we write $\mathfrak{S} \to_{\mathsf{R}} \mathfrak{S}'$ or simply $\mathfrak{S} \to_S \mathfrak{S}'$ if it is not relevant which of the rules of the tableau S was applied.* ■

As usual, we denote the *reflexive-transitive closure* of $\to_S$ by $\overset{*}{\to}_S$. The rules are applied to the S-state until it becomes saturated; that is, until no rule can be applied anymore. At that point, we can use the set of clashes to decide the property: the axiomatised input is accepted (in other words, belongs to the property decided by the algorithm) if and only if it contains an element of $\mathcal{C}$. Returning to subsumption of $\mathcal{HL}$ concept names, $\mathbf{A}$ is subsumed by $\mathbf{B}$ w.r.t. $\mathcal{T}$ iff the saturated S-state found in this way contains the pair $(\mathbf{A}, \mathbf{B})$. Thus, in our tableau setting, the set of clashes consists of all sets of the form $\{(\mathbf{A}, \mathbf{B}), \mathbf{A} \sqsubseteq^? \mathbf{B}\}$, where $\mathbf{A}, \mathbf{B}$ are concept names.

Definition 3.7 (Saturated state, clash). *An S-state $\mathfrak{S} = (A, \mathcal{T})$ is called* saturated *iff there is no $\mathfrak{S}'$ such that $\mathfrak{S} \to_S \mathfrak{S}'$. It* contains a clash *iff there is a set $C \in \mathcal{C}$ such that $C \subseteq A$.* ■

For a ground tableau to correctly decide a c-property it needs first to be a terminating procedure and second to adequately find a clash in the state found after termination, as expressed in the following definition.

Definition 3.8 (Correctness). *Let $\mathcal{P}$ be a c-property on axiomatised inputs for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$, and S a ground tableau for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$. We say that S is* correct for $\mathcal{P}$ *if the following holds for every axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$ for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$:*

1. *S terminates on Γ; that is, there exists no infinite chain of rule applications $\mathfrak{S}_0 \to_S \mathfrak{S}_1 \to_S \ldots$ starting with $\mathfrak{S}_0 = \Gamma^S$.*

2. *For every chain of rule applications $\mathfrak{S}_0 \overset{*}{\to}_S \mathfrak{S}_n$ such that $\mathfrak{S}_0 = \Gamma^S$ and $\mathfrak{S}_n$ is saturated, we have $\Gamma \in \mathcal{P}$ iff $\mathfrak{S}_n$ contains a clash.*

■

The second condition for correctness given in this definition might seem like a strong restriction, since it forces the algorithm to yield the same result regardless of the order in which rules are applied, making it sufficient to test only one such order to decide the property. Actually, the fact that the order in which rules are applied is irrelevant for the presence or absence of a clash is hardcoded in our notion of ground tableau, as shown in the next proposition. This means that although the order in which rules are applied can be seen as a source of non-determinism, it is of the do-not-care kind, and hence we need not worry about it.

Proposition 3.9. *Let Γ be an axiomatised input and $\mathfrak{S}_0 = \Gamma^S$. If $\mathfrak{S}$ and $\mathfrak{S}'$ are saturated S-states such that $\mathfrak{S}_0 \xrightarrow{*}_S \mathfrak{S}$ and $\mathfrak{S}_0 \xrightarrow{*}_S \mathfrak{S}'$, then $\mathfrak{S}$ contains a clash iff $\mathfrak{S}'$ contains a clash.*

A correct tableau can be used to decide whether a given axiomatised input belongs to a property or not. We proceed now to show how it can be extended to an algorithm that computes a pinpointing formula. Recall the assumption made for the definition of the pinpointing formula that every axiom $t \in \mathcal{T}$ is labeled with a unique propositional variable $\mathsf{lab}(t)$, and the set of all propositional variables labeling an axiom in $\mathcal{T}$ is denoted by $\mathsf{lab}(\mathcal{T})$.

Given an axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$, the modified algorithm also works on sets of S-states, but now every assertion a occurring in the first component of an S-state is equipped with a label $\mathsf{lab}(a)$, which is a monotone Boolean formula over $\mathsf{lab}(\mathcal{T})$. We call such S-states *labeled S-states*. Intuitively, the label of an assertion expresses the axioms that are necessary to produce it. Thus, in the initial S-state $(A, \mathcal{T}) = (\mathcal{I}, \mathcal{T})^S$, an assertion $a \in A$ is labeled with $\top$ if $a \in \mathcal{I}^S$ and with $\bigvee_{\{t \in \mathcal{T} \mid a \in t^S\}} \mathsf{lab}(t)$ otherwise. The intuition of these labels is that, if $a \in \mathcal{I}^S$, then the assertion a will be produced by the tableaux algorithm, regardless of the axioms included in the input. Otherwise, the label expresses which axioms are the responsible for its appearance in the initial state.

For instance, consider again our tableau for subsumption w.r.t. $\mathcal{HL}$ TBoxes and the axiomatised input $\Gamma = ((\mathbf{A}, \mathbf{B}), \mathcal{T})$, where $\mathcal{T}$ has only the axioms in (3.1). The initial function maps Γ to the S-state having the following set of labeled assertions:[11]

$$\{\mathbf{A} \sqsubseteq^? \mathbf{B}^\top, (\mathbf{A}, \mathbf{A})^{\mathrm{ax}_1 \vee \mathrm{ax}_2}, (\mathbf{B}, \mathbf{B})^{\mathrm{ax}_4}, (\mathbf{C}, \mathbf{C})^{\mathrm{ax}_1 \vee \mathrm{ax}_3 \vee \mathrm{ax}_4}, (\mathbf{D}, \mathbf{D})^{\mathrm{ax}_2 \vee \mathrm{ax}_3}\}. \quad (3.3)$$

The definition of rule application must also take the labels of assertions and axioms into account. Let A be a set of labeled assertions and ψ a monotone Boolean formula. We say that the (unlabeled) assertion a is *ψ-insertable into A* if either (i) $a \notin A$, or (ii) $a \in A$, with $\mathsf{lab}(a) = \phi$, but $\psi \not\models \phi$. Given a set B of assertions and a set A of labeled assertions, the set of *ψ-insertable elements of B into A* is defined as $\mathsf{ins}_\psi(B, A) := \{b \in B \mid b \text{ is } \psi\text{-insertable into } A\}$.[12] By ψ-inserting these insertable elements into A, we obtain the new set of labeled assertions given by:

$$A \uplus_\psi B := A \cup \mathsf{ins}_\psi(B, A),$$

where each assertion $a \in A \setminus \mathsf{ins}_\psi(B, A)$ keeps its old label $\mathsf{lab}(a)$, each assertion in $\mathsf{ins}_\psi(B, A) \setminus A$ gets label ψ, and each assertion $b \in A \cap \mathsf{ins}_\psi(B, A)$ gets the new label $\psi \vee \mathsf{lab}(b)$.

Definition 3.10 (Pinpointing rule application). *Let $\mathfrak{S} = (A, \mathcal{T})$ be a labeled S-state and $\mathsf{R} : (B_0, \mathcal{S}) \to B$ a rule.* R *is* pinpointing applicable to $\mathfrak{S}$ *if (i) $\mathcal{S} \subseteq \mathcal{T}$, (ii) $B_0 \subseteq A$, and (iii) $\mathsf{ins}_\psi(B, A) \neq \emptyset$, where $\psi := \bigwedge_{b \in B_0} \mathsf{lab}(b) \wedge \bigwedge_{s \in \mathcal{S}} \mathsf{lab}(s)$.*

[11] For simplicity, we sometimes represent the labels of assertions by means of superscripts; i.e., if a is an assertion, then a^ϕ denotes the labeled assertion where $\mathsf{lab}(a) = \phi$.

[12] Notice here that the set B contains *unlabeled* assertions. This is consistent with the fact that rules of a tableau use only unlabeled assertions; the labels are treated by a modified rule application.

Given a labeled S-state $\mathfrak{S} = (A, \mathcal{T})$ *to which the rule* R *is pinpointing applicable, the* pinpointing application of R to $\mathfrak{S}$ *yields the new S-state* $(A \uplus_\psi B, \mathcal{T})$*, where the formula* ψ *is defined as above.*

If $\mathfrak{S}'$ *is obtained from* $\mathfrak{S}$ *by the pinpointing application of the rule* R*, then we write* $\mathfrak{S} \to_{\mathsf{R}^{pin}} \mathfrak{S}'$*, or simply* $\mathfrak{S} \to_{S^{pin}} \mathfrak{S}'$ *if it is not relevant which of the rules of the tableau* S *was applied. A labeled S-state* $\mathfrak{S}$ *is* pinpointing saturated *if there is no* $\mathfrak{S}'$ *such that* $\mathfrak{S} \to_{S^{pin}} \mathfrak{S}'$*.* ■

Returning to our example, we show how pinpointing rule applications modify the labeled state Γ^S in (3.3). The assertion $(\mathbf{A}, \mathbf{A})$ along with axiom ax_2 can trigger the rule hl in order to add the assertion $(\mathbf{A}, \mathbf{D})$ to this state, with the label $(\mathrm{ax}_1 \vee \mathrm{ax}_2) \wedge \mathrm{ax}_2$. For the sake of readability, we will simplify this formula. Hence, $\mathsf{lab}((\mathbf{A}, \mathbf{D})) = \mathrm{ax}_2$. This newly generated assertion can now be used in combination with axiom ax_3 to add the assertion $(\mathbf{A}, \mathbf{C})$, which will have as label the conjunction of $\mathsf{lab}((\mathbf{A}, \mathbf{D}))$ and ax_3; i.e., $\mathsf{lab}((\mathbf{A}, \mathbf{C})) = \mathrm{ax}_2 \wedge \mathrm{ax}_3$. Notice now that the assertion $(\mathbf{A}, \mathbf{A})$ can also trigger a rule application by means of axiom ax_1. Since this rule application would only add the assertion $(\mathbf{A}, \mathbf{C})$ that is already present in the current S-state, it would be disallowed in the original tableau sense. However, since this shows an alternate way to obtain the same assertion, it needs to be allowed by pinpointing rule application, as is the case because $(\mathrm{ax}_1 \vee \mathrm{ax}_2) \wedge \mathrm{ax}_1 \not\models \mathsf{lab}((\mathbf{A}, \mathbf{C}))$. When the rule is pinpointing applied, no assertion is added to the set, but the label of $(\mathbf{A}, \mathbf{C})$ is changed to $((\mathrm{ax}_1 \vee \mathrm{ax}_2) \wedge \mathrm{ax}_1) \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3)$, or, equivalently, $\mathrm{ax}_1 \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3)$. Finally, the assertions $(\mathbf{A}, \mathbf{C})$ and $(\mathbf{A}, \mathbf{D})$ can be used along axiom ax_4 to introduce the assertion $(\mathbf{A}, \mathbf{B})$, whose label is given by $\mathsf{lab}((\mathbf{A}, \mathbf{C})) \wedge \mathsf{lab}((\mathbf{A}, \mathbf{D})) \wedge \mathrm{ax}_4$; that is,

$$(\mathrm{ax}_1 \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3)) \wedge \mathrm{ax}_2 \wedge \mathrm{ax}_4.$$

Recall now that the original tableau decides the property by verifying the presence of a clash. In the subsumption example, the clash consists of the set of assertions $\{\mathbf{A} \sqsubseteq^? \mathbf{B}, (\mathbf{A}, \mathbf{B})\}$. The conjunction of the labels of both assertions tells us which axioms are necessary for the clash to exist. In this case, the so-called clash formula is $\top \wedge (\mathrm{ax}_1 \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3)) \wedge \mathrm{ax}_2 \wedge \mathrm{ax}_4$. Clearly, it is equivalent to the pinpointing formula $(\mathrm{ax}_1 \vee \mathrm{ax}_3) \wedge \mathrm{ax}_2 \wedge \mathrm{ax}_4$ that was presented in Section 3.1. In general, consider a chain of pinpointing rule applications $\mathfrak{S}_0 \to_{S^{pin}} \ldots \to_{S^{pin}} \mathfrak{S}_n$ such that $\mathfrak{S}_0 = \Gamma^S$ for an axiomatised input Γ and $\mathfrak{S}_n$ is pinpointing saturated. The label of an assertion in $\mathfrak{S}_n$ expresses which axioms are needed to obtain this assertion. A clash in $\mathfrak{S}_n$ depends on the joint presence of certain assertions. Thus, we define the label of the clash as the conjunction of the labels of these assertions. Since it is enough to have just one clash in $\mathfrak{S}_n$, the labels of different clashes in this state are combined disjunctively.

Definition 3.11 (Clash set, clash formula). *Let* $\mathfrak{S} = (A, \mathcal{T})$ *be a labeled S-state and* $A' \subseteq A$*. Then* A' *is a* clash set in $\mathfrak{S}$ *if* $A' \in \mathcal{C}$*. The* label *of this clash set is* $\psi_{A'} := \bigwedge_{a \in A'} \mathsf{lab}(a)$*.*

Let $\mathfrak{S}$ *be a labeled S-state. The* clash formula induced by $\mathfrak{S}$ *is defined as*

$$\psi_{\mathfrak{S}} := \bigvee_{A' \text{ clash set in } \mathfrak{S}} \psi_{A'}.$$

■

Recall that, given a set $\mathcal{T}$ of labeled axioms, a propositional valuation $\mathcal{V}$ induces the subset $\mathcal{T}_\mathcal{V} := \{t \in \mathcal{T} \mid \mathsf{lab}(t) \in \mathcal{V}\}$ of $\mathcal{T}$. Similarly, for a set A of labeled assertions, the valuation $\mathcal{V}$ induces the subset $A_\mathcal{V} := \{a \in A \mid \mathcal{V} \text{ satisfies } \mathsf{lab}(a)\}$. Given a labeled S-state $\mathfrak{S} = (A, \mathcal{T})$ we define its *$\mathcal{V}$-projection* as $\mathcal{V}(\mathfrak{S}) := (A_\mathcal{V}, \mathcal{T}_\mathcal{V})$. The following lemma is an easy consequence of the definition of the clash formula:

Lemma 3.12. *Let $\mathfrak{S}$ be a labeled S-state and $\mathcal{V}$ a propositional valuation. Then we have that $\mathcal{V}$ satisfies $\psi_\mathfrak{S}$ iff $\mathcal{V}(\mathfrak{S})$ contains a clash.*

There is also a close connection between pinpointing saturatedness of a labeled S-state and saturatedness of its projection:

Lemma 3.13. *Let $\mathfrak{S}$ be a labeled S-state and $\mathcal{V}$ a propositional valuation. If $\mathfrak{S}$ is pinpointing saturated, then $\mathcal{V}(\mathfrak{S})$ is saturated.*

Given a tableau that is correct for a property $\mathcal{P}$, its pinpointing extension is correct in the sense that the clash formula induced by the pinpointing saturated set computed by a terminating chain of pinpointing rule applications is indeed a pinpointing formula for $\mathcal{P}$ and the axiomatised input.

Theorem 3.14 (Correctness of pinpointing). *Let $\mathcal{P}$ be a c-property on axiomatised inputs for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$, and S a correct tableau for $\mathcal{P}$. Then the following holds for every axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$ for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$:*

> *For every chain of rule applications $\mathfrak{S}_0 \to_{S^{pin}} \ldots \to_{S^{pin}} \mathfrak{S}_n$ such that $\mathfrak{S}_0 = \Gamma^S$ and $\mathfrak{S}_n$ is pinpointing saturated, the clash formula $\psi_{\mathfrak{S}_n}$ induced by $\mathfrak{S}_n$ is a pinpointing formula for $\mathcal{P}$ and Γ.*

In this section we have defined ground tableaux and shown how each of them can be extended into an algorithm that computes a pinpointing formula for a given property and axiomatised input. While this framework suffices to deal with the very inexpressive logic $\mathcal{HL}$, it lacks the expressivity for dealing with two phenomena that appear already in the algorithm for deciding consistency of $\mathcal{ALC}$ ABoxes (Section 2.3.2); namely, non-determinism, and assertions with an internal structure. The next section extends the ideas of ground tableaux, defining a more general notion that can successfully deal with these phenomena.

3.3 Pinpointing in General Tableaux

In this section we follow the same path of Section 3.2: we first formalise the notion of a tableau-like decision procedure, and then show how it can be modified to obtain an algorithm that computes a pinpointing formula. The structure of these two steps follows the same main ideas used in the previous section, but in a more general setting that can deal both with non-deterministic rules, and with assertions having an internal structure. For this part, we will use the algorithm described in Section 2.3.2, in

which both phenomena appear, as an intuitive basis for the notions that will be introduced. Notice, nonetheless, that the c-property decided by that algorithm is actually *in*consistency; analogously, in the algorithms presented in Sections 2.3.3 to 2.3.5 we will be interested in *un*satisfiability of concepts.

With respect to non-determinism, consider the rule $\mathsf{alc}_\sqcup$ shown in Figure 2.3. When our model candidate contains a concept of the form $C \sqcup D$, then we need to choose (do-not-know) non-deterministically which of the disjuncts to use to extend it. In order to represent this, the rules in a general tableau will have on the right-hand side a finite set of sets of assertions, rather than simply a set of assertions as in the previous section. More formally, a rule is of the form $(B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$, where $B_0, B_1, \ldots, B_m$ are finite sets of assertions and $\mathcal{S}$ is a finite set of axioms. Thus, ignoring for the moment the variables, the $\mathsf{alc}_\sqcup$ rule could be represented in this setting as

$$\mathsf{alc}_\sqcup : (\{C \sqcup D\}, \emptyset) \to \{\{C\}, \{D\}\}.$$

Instead of dealing only with S-states, the decision algorithm will operate over sets of S-states, where the application of a rule R substitutes one of these S-states with as many S-states as there are elements in the right-hand side of R. Basically, each S-state in the set represents one of the non-deterministic options that needs to be verified. For instance, if we have the singleton set $\{(\{C \sqcup D\}, \emptyset)\}$, an application of the rule $\mathsf{alc}_\sqcup$ will lead to the set $\{(\{C \sqcup D, C\}, \emptyset), (\{C \sqcup D, D\}, \emptyset)\}$, where the first element expresses the path where the concept C is selected to be satisfied, and the second, that in which D is the satisfied concept.

Regarding the structure of assertions, notice the tableaux-based algorithms for $\mathcal{ALC}$ use as assertions not merely concept terms, but have individuals associated with them; i.e., the assertions have the form $C(a)$ or $r(a, b)$, with C a concept name, r a role name and a, b two individuals. In general, we have *structured assertions* of the form $P(a_1, \ldots, a_k)$, where P is a k-ary predicate and $a_1, \ldots, a_k$ are constants. Naturally it is not necessary to define a rule for each specific constant; we instead allow variables to act as placeholders for them.

Furthermore, rules should be able to create new constants. For example, consider the rule $\mathsf{alc}_\exists$ appearing also in Figure 2.3. The application of this rule requires us to create a new individual name. Such a rule will be written in the general tableaux setting as

$$\mathsf{alc}_\exists : (\{(\exists r.C)(x)\}, \emptyset) \to \{\{r(x, y), C(y)\}\}.$$

In order to apply this rule to an S-state, we need to appropriately replace the variables in the left-hand side by constants. The variable y is what will be called a *fresh variable*; that is, one that appears only on the right-hand side of a rule. Fresh variables are replaced by constants that do not appear in the S-state to which the rule is being applied. In order to avoid that such a rule is applied indefinitely, creating new individuals with each application, the applicability condition needs to be modified to check whether it is possible to replace the fresh variables by old constants to obtain assertions in the current S-state.

We begin by formalising all these notions. In the following we will use $\mathbf{V}$ and $\mathbf{D}$ to denote countably infinite sets whose elements are called *variables*; and *constants*,

respectively. A *signature* Σ is a set of predicate symbols, where each predicate $P \in \Sigma$ is associated to a (fixed) arity. A Σ-assertion is of the form $P(a_1, \ldots, a_n)$, where $P \in \Sigma$ is a predicate of arity n and $a_1, \ldots, a_n$ are constants from $\mathbf{D}$. Likewise, a Σ-pattern is of the form $P(x_1, \ldots, x_n)$ where $P \in \Sigma$ is an n-ary predicate and $x_1, \ldots, x_n \in \mathbf{V}$. Whenever the signature is clear from the context, we will often use it implicitly and simply say *pattern* or *assertion*. Given a set A of assertions, we will use the expression $\mathsf{cons}(A)$ to denote the set of constants appearing in A. In the same fashion, $\mathsf{var}(B)$ denotes the set of variables that appear in a set B of patterns.

A *substitution* is a mapping $\sigma : V \to \mathbf{D}$, where $V \subseteq \mathbf{V}$ is a finite set of variables. In this case we say that σ is a substitution *on* V. If B is a set of patterns such that $\mathsf{var}(B) \subseteq V$, then $B\sigma$ denotes the set of assertions obtained from B by replacing each variable by its image under σ. If σ is a substitution on V and θ a substitution on V' such that $V \subseteq V'$ and $\theta(x) = \sigma(x)$ for all $x \in V$, then we say that θ *extends* σ.

We are ready now to describe the notion of *general tableaux*, which generalises the ideas of ground tableaux presented in the previous section by allowing non-deterministic rules and structured assertions.

Definition 3.15 (General tableau). *Let $\mathfrak{I}$ be a set of inputs and $\mathscr{P}_{admis}(\mathfrak{T})$ an admissible set of sets of elements in $\mathfrak{T}$. A* general tableau *for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$ is a tuple $S = (\Sigma, \cdot^S, \mathcal{R}, \mathcal{C})$ where*

- *Σ is a signature;*
- *$\cdot^S$ is a function that maps every $\mathcal{I} \in \mathfrak{I}$ to a finite set of finite sets of Σ-assertions and every $t \in \mathfrak{T}$ to a finite set of Σ-assertions;*
- *$\mathcal{R}$ is a set of* rules *of the form $(B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$ where $B_0, \ldots, B_m$ are finite sets of Σ-patterns and $\mathcal{S}$ is a finite set of axioms;*
- *$\mathcal{C}$ is a set of finite sets of Σ-patterns, called* clashes.

■

As for ground tableaux, we extend the function $\cdot^S$ to axiomatised inputs by setting

$$(\mathcal{I}, \mathcal{T})^S = \{(A \cup \bigcup_{t \in \mathcal{T}} t^S, \mathcal{T}) \mid A \in \mathcal{I}^S\}.$$

Notice that in this case, given an axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$, Γ^S does not define a single S-state, but rather a whole set of them. Intuitively, each set represents a non-deterministic choice for the algorithm to begin to iterate with. In order to decide a property affirmatively, each of these sets needs to produce a clash. We need to extend the notion of a rule application too. In this case, we cannot just extend the only S-state; instead, rules modify the current set of S-states $\mathcal{M}$. Each rule application selects an S-state $\mathfrak{S}$ from $\mathcal{M}$ and replaces it by finitely many new S-states $\mathfrak{S}_1, \ldots, \mathfrak{S}_m$ that extend the first component of $\mathfrak{S}$.

Definition 3.16 (Rule application). *Suppose we have an S-state* $\mathfrak{S} = (A, \mathcal{T})$*, a rule* $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\} \in \mathcal{R}$ *and a substitution* ρ *on* $\mathsf{var}(B_0)$*. We say that* R *is* applicable to $\mathfrak{S}$ with ρ *if the following three conditions are satisfied: (i)* $\mathcal{S} \subseteq \mathcal{T}$*, (ii)* $B_0\rho \subseteq A$*, and (iii) for every* $i, 1 \leq i \leq m$ *and every substitution* ρ' *on* $\mathsf{var}(B_0 \cup B_i)$ *extending* ρ *it holds that* $B_i\rho' \not\subseteq A$*.*

Given a set of S-states $\mathcal{M}$*, an S-state* $\mathfrak{S} = (A, \mathcal{T}) \in \mathcal{M}$ *and a rule* R*, if* R *is applicable to* $\mathfrak{S}$ *with substitution* ρ*, then the* application of R to $\mathfrak{S}$ with ρ in $\mathcal{M}$ *yields the new set of S-states* $\mathcal{M}' = (\mathcal{M} \setminus \{\mathfrak{S}\}) \cup \{(A \cup B_i\sigma, \mathcal{T}) \mid 1 \leq i \leq m\}$*, where* σ *is a substitution on the variables appearing in* R *that extends* ρ *and maps the fresh variables of* R *to distinct new constants; i.e., constants that do not appear in A.*

If $\mathcal{M}'$ *is obtained from* $\mathcal{M}$ *by the application of the rule* R*, then we write* $\mathcal{M} \to_{\mathsf{R}} \mathcal{M}'$ *or simply* $\mathcal{M} \to_S \mathcal{M}'$ *if it is not relevant which rule of the tableau S is applied.* ■

The conditions of applicability ensure that the same rule R cannot be applied indefinitely using the same substitution ρ, but it may well be the case that the new added constants trigger repeated rule applications, yielding a non-terminating procedure. Let us for a moment assume that this is not the case, and we can reach a set of S-states where no rule can be applied. When no rules are applicable to $\mathcal{M}$, we check for clashes in each of the states belonging to $\mathcal{M}$. The decision made by the algorithm will depend on the presence or absence of these clashes.

Definition 3.17 (Saturated, clash). *The set of S-states* $\mathcal{M}$ *is called* saturated *if there is no* $\mathcal{M}'$ *such that* $\mathcal{M} \to_S \mathcal{M}'$*.*

The S-state $\mathfrak{S} = (A, \mathcal{T})$ contains a clash *if there is a set of patterns* $C \in \mathcal{C}$ *and a substitution* ρ *on* $\mathsf{var}(C)$ *such that* $C\rho \subseteq A$*; the set of S-states* $\mathcal{M}$ *is* full of clashes *if each of its elements contains a clash.* ■

To decide whether a property holds, we need to check at the saturated set of S-states reached by the application of the tableaux rules. In Section 2.3.2, we see that the input ABox is inconsistent if and only if all the states in this set contain a clash. The same condition appears in the subsequent section, for deciding unsatisfiability of a concept with respect to an acyclic TBox. Thus, in a general tableau, we will say that the axiomatic input belongs to a property if after finitely many rule applications we reach a saturated set of states that is full of clashes.

Definition 3.18 (Correctness). *Let* $\mathcal{P}$ *be a c-property on axiomatised inputs for* $\mathfrak{I}$ *and* $\mathscr{P}_{admis}(\mathfrak{T})$*, and S a general tableau for* $\mathfrak{I}$ *and* $\mathscr{P}_{admis}(\mathfrak{T})$*. We say that S is* correct for $\mathcal{P}$ *if the following holds for every axiomatised input* $\Gamma = (\mathcal{I}, \mathcal{T})$ *for* $\mathfrak{I}$ *and* $\mathscr{P}_{admis}(\mathfrak{T})$*:*

1. *S terminates on* Γ*; that is, there exists no infinite chain of rule applications* $\mathcal{M}_0 \to_S \mathcal{M}_1 \to_S \ldots$ *starting with* $\mathcal{M}_0 = \Gamma^S$*.*
2. *For every chain of rule applications* $\mathcal{M}_0 \overset{*}{\to}_S \mathcal{M}_n$ *such that* $\mathcal{M}_0 = \Gamma^S$ *and* $\mathcal{M}_n$ *is saturated, we have* $\Gamma \in \mathcal{P}$ *iff* $\mathcal{M}_n$ *is full of clashes.*

■

It is easy to see that ground tableaux are indeed a special case of general tableaux, in which the signature contains only nullary predicates and all the rules are *deterministic*; that is, they have a singleton set on their right-hand side. Even in the more general setting of this section, we can show a result analogous to Proposition 3.9 stating that the rule application order is irrelevant to the decision made by the tableau.

Proposition 3.19. *Let Γ be an axiomatised input and $\mathcal{M}_0 = \Gamma^S$. If $\mathcal{M}$ and $\mathcal{M}'$ are saturated sets of S-states such that $\mathcal{M}_0 \overset{*}{\to}_S \mathcal{M}$ and $\mathcal{M}_0 \overset{*}{\to}_S \mathcal{M}'$, then $\mathcal{M}$ is full of clashes iff $\mathcal{M}'$ is full of clashes.*

This proposition actually follows directly from Lemma 3.31, and hence we delay its proof until there. A direct proof of the proposition would be almost identical to that presented for Lemma 3.31.

Given a general tableau $S = (\Sigma, \cdot^S, \mathcal{R}, \mathcal{C})$ that is correct of a property $\mathcal{P}$, we show how the algorithm for deciding $\mathcal{P}$ induced by S can be modified into an algorithm that computes a pinpointing formula for $\mathcal{P}$. As in the ground case, the modified algorithm works in a fashion similar to the original tableau, based on S-states, but now every assertion a occurring in the assertion component of an S-state is equipped with a label $\mathsf{lab}(a)$ which is a monotone Boolean formula over $\mathsf{lab}(\mathcal{T})$.

The assertions appearing in an initial state are labeled in the same way as in the previous section; that is, given an initial S-state $(A, \mathcal{T}) \in (\mathcal{I}, \mathcal{T})^S$, an assertion $a \in A$ is labeled with $\top$ if $a \in \mathcal{I}^S$ and with $\bigvee_{\{t \in \mathcal{T} \mid a \in t^S\}} \mathsf{lab}(t)$ otherwise.

Definition 3.20 (Pinpointing rule application). *Assume there is a labeled S-state $\mathfrak{S} = (A, \mathcal{T})$, a rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$, and a substitution ρ on $\mathsf{var}(B_0)$. This rule is* pinpointing applicable to $\mathfrak{S}$ with ρ *if the following conditions hold: (i) $\mathcal{S} \subseteq \mathcal{T}$, (ii) $B_0\rho \subseteq A$, and (iii) for every $i, 1 \leq i \leq m$, and every substitution ρ' on $\mathsf{var}(B_0 \cup B_i)$ extending ρ we have $\mathsf{ins}_\psi(B_i\rho', A) \neq \emptyset$, where*

$$\psi = \bigwedge_{b \in B_0} \mathsf{lab}(b\rho) \wedge \bigwedge_{s \in \mathcal{S}} \mathsf{lab}(s). \tag{3.4}$$

Given a set of labeled S-states $\mathcal{M}$ and a labeled S-state $\mathfrak{S} = (A, \mathcal{T}) \in \mathcal{M}$ to which the rule R is pinpointing applicable with substitution ρ, the pinpointing application of R to $\mathfrak{S}$ with ρ in $\mathcal{M}$ *yields the new set of labeled states*

$$\mathcal{M}' = (\mathcal{M} \setminus \{\mathfrak{S}\}) \cup \{(A \Cup_\psi B_i\sigma, \mathcal{T}) \mid 1 \leq i \leq m\},$$

where the formula ψ is defined as in Equation (3.4) and σ is a substitution on the variables occurring in R that extends ρ and maps the fresh variables of R to distinct new constants.

If $\mathcal{M}'$ is obtained from $\mathcal{M}$ by the pinpointing application of R, then we write $\mathcal{M} \to_{\mathsf{R}^{pin}} \mathcal{M}'$, or simply $\mathcal{M} \to_{S^{pin}} \mathcal{M}'$ if the rule applied is not relevant. A set of labeled S-states $\mathcal{M}$ is called pinpointing saturated *if there is no $\mathcal{M}'$ such that $\mathcal{M} \to_{S^{pin}} \mathcal{M}'$.* ■

Consider a chain of pinpointing rule applications $\mathcal{M}_0 \to_{S^{pin}} \ldots \to_{S^{pin}} \mathcal{M}_n$ such that $\mathcal{M}_0 = \Gamma^S$ for an axiomatised input Γ and $\mathcal{M}_n$ is pinpointing saturated. The

label of an assertion in $\mathcal{M}_n$ expresses which axioms are needed to obtain said assertion. Thus, we define the label of a clash as the conjunction of the labels of all the assertions appearing in it. Since it is enough to have just one clash per S-state $\mathfrak{S}$, the labels of different clashes in $\mathfrak{S}$ are combined disjunctively. Finally, since we need a clash in every S-state of $\mathcal{M}_n$, the formulae obtained from the single S-states are again conjoined.

Definition 3.21 (Clash set, clash formula). *Let $\mathfrak{S} = (A, \mathcal{T})$ be a labeled S-state and $A' \subseteq A$. Then A' is a* clash set in $\mathfrak{S}$ *if there is a clash $C \in \mathcal{C}$ and a substitution ρ on $\mathsf{var}(C)$ such that $A' = C\rho$. The* label *of this clash set is given by $\psi_{A'} = \bigwedge_{a \in A'} \mathsf{lab}(a)$.*

Let $\mathcal{M} = \{\mathfrak{S}_1, \ldots, \mathfrak{S}_n\}$ be a set of labeled S-states. The clash formula induced by *$\mathcal{M}$ is defined as*

$$\psi_{\mathcal{M}} = \bigwedge_{i=1}^{n} \bigvee_{A' \text{ clash set in } \mathfrak{S}_i} \psi_{A'}$$

■

In the previous section we defined the $\mathcal{V}$-projection of a labeled S-state $\mathfrak{S} = (A, \mathcal{T})$ as $\mathcal{V}(\mathfrak{S}) = (A_{\mathcal{V}}, \mathcal{T}_{\mathcal{V}})$. We now extend this notion to sets of S-states $\mathcal{M}$ in the obvious way: $\mathcal{V}(\mathcal{M}) = \{\mathcal{V}(\mathfrak{S}) \mid \mathfrak{S} \in \mathcal{M}\}$.

Lemma 3.22. *Let $\mathcal{M}$ be a finite set of labeled S-states and $\mathcal{V}$ a propositional valuation. Then $\mathcal{V}$ satisfies $\psi_{\mathcal{M}}$ iff $\mathcal{V}(\mathcal{M})$ is full of clashes.*

Proof. We will prove the *if* direction first. For that, assume that $\mathcal{V}(\mathcal{M})$ is full of clashes. We know then that for every S-state $\mathfrak{S}_i \in \mathcal{M}$ the projection $\mathcal{V}(\mathfrak{S}_i)$ contains a clash. Thus, for every i there is a clash set A_i in $\mathfrak{S}_i$ such that $\mathsf{lab}(a)$ is satisfied by $\mathcal{V}$ for every assertion $a \in A_i$. This means that $\mathcal{V}$ satisfies ψ_{A_i}, and hence $\mathcal{V}$ also satisfies the formula

$$\bigvee_{A' \text{ clash set in } \mathfrak{S}_i} \psi_{A'}.$$

Since this is true for every $\mathfrak{S}_i \in \mathcal{M}$, the valuation $\mathcal{V}$ satisfies also the clash formula $\psi_{\mathcal{M}}$.

Conversely, assume for the *only if* direction that $\mathcal{V}(\mathcal{M})$ is not full of clashes; i.e., there exists a $\mathfrak{S}_i \in \mathcal{M}$ such that $\mathcal{V}(\mathfrak{S}_i)$ does not contain a clash. For this to happen it must be the case that for every clash set $A' \in \mathfrak{S}_i$ there is an assertion $a \in A'$ such that $\mathcal{V}$ does not satisfy $\mathsf{lab}(a)$. Consequently, $\mathcal{V}$ does not satisfy the label $\psi_{A'}$ of any of the clash sets A' in $\mathfrak{S}_i$, and thus this valuation cannot satisfy the disjunction of such labels. This shows that $\mathcal{V}$ does not satisfy the clash formula. □

There is also a close connection between the pinpointing saturatedness of a set of labeled S-states and the saturatedness of its projection.

Lemma 3.23. *Let $\mathcal{M}$ be a finite set of labeled S-states and $\mathcal{V}$ a propositional valuation. If $\mathcal{M}$ is pinpointing saturated, then $\mathcal{V}(\mathcal{M})$ is saturated.*

Proof. Suppose that $\mathcal{V}(\mathcal{M})$ is not saturated; in other words, that there is an S-state $\mathfrak{S} = (A, \mathcal{T}) \in \mathcal{M}$ and a rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$ such that R is applicable to $\mathcal{V}(\mathfrak{S})$ with substitution ρ. We will show that R is pinpointing applicable to $\mathfrak{S}$ with the same substitution ρ, and hence $\mathcal{M}$ is not pinpointing saturated.

By Definition 3.6, since R is applicable to $\mathcal{V}(\mathfrak{S})$ with substitution ρ, we know that (i) $\mathcal{S} \subseteq \mathcal{T}_\mathcal{V}$, (ii) $B_0\rho \subseteq A_\mathcal{V}$, and (iii) for every $i, 1 \leq i \leq m$ and every substitution ρ' on $\mathsf{var}(B_0 \cup B_i)$ extending ρ, it holds that $B_i\rho' \not\subseteq A_\mathcal{V}$. Since $\mathcal{S} \subseteq \mathcal{T}_\mathcal{V} \subseteq \mathcal{T}$ and $B_0\rho \subseteq A_\mathcal{V} \subseteq A$, the first two conditions of the definition of pinpointing applicability of rules (Definition 3.20) are satisfied. We need now only to show that the third condition is also satisfied. Consider an arbitrary but fixed i and a substitution ρ' on $\mathsf{var}(B_0 \cup B_i)$ extending ρ. We must show that $\mathsf{ins}_\psi(B_i\rho', A) \neq \emptyset$, where

$$\psi = \bigwedge_{b \in B_0} \mathsf{lab}(b\rho) \wedge \bigwedge_{s \in \mathcal{S}} \mathsf{lab}(s).$$

Notice that $\mathcal{S} \subseteq \mathcal{T}_\mathcal{V}$ and $B_0\rho \subseteq A_\mathcal{V}$ imply that $\mathcal{V}$ satisfies ψ. Since $B_i\rho' \not\subseteq A_\mathcal{V}$, there must exist a $b \in B_i$ such that $b\rho' \notin A_\mathcal{V}$. This means that either $b\rho' \notin A$ or $\mathcal{V}$ does not satisfy $\mathsf{lab}(b\rho')$. In the first case, $b\rho'$ is clearly ψ-insertable into A; in the second, it holds that $\psi \not\models \mathsf{lab}(b\rho')$ since $\mathcal{V}$ satisfies ψ, and thus $b\rho'$ is again ψ-insertable into A. Hence, $\mathsf{ins}_\psi(B_i\rho', A) \neq \emptyset$, which implies that R is pinpointing applicable to $\mathfrak{S}$ with substitution ρ. □

Given a tableau that is correct for a property $\mathcal{P}$, its pinpointing extension is correct in the sense that the clash formula induced by the pinpointing saturated set computed by a terminating chain of pinpointing rule applications is indeed a pinpointing formula for $\mathcal{P}$ and the input.

Theorem 3.24. *Let $\mathcal{P}$ be a c-property on axiomatised inputs over $\mathcal{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$, and S a correct tableau for $\mathcal{P}$. Then, for every axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$ over $\mathcal{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$ it holds that*

> *For every chain of rule applications $\mathcal{M}_0 \to_{S^{pin}} \ldots \to_{S^{pin}} \mathcal{M}_n$ such that $\mathcal{M}_0 = \Gamma^S$ and $\mathcal{M}_n$ is pinpointing saturated, the clash formula $\psi_{\mathcal{M}_n}$ induced by $\mathcal{M}_n$ is a pinpointing formula for $\mathcal{P}$ and Γ.*

We will prove this theorem by *projecting* chains of pinpointing rule applications to chains of tableau rule applications as in Definition 3.16. Unfortunately such a projection cannot be done in a straightforward manner since in general a pinpointing rule application $\mathcal{M} \to_{S^{pin}} \mathcal{M}'$ does not imply that $\mathcal{V}(\mathcal{M}) \to_S \mathcal{V}(\mathcal{M}')$. There are two possible reasons for this. First, it could be the case that the assertions and axioms to which the pinpointing rule was applied in $\mathcal{M}$ are not present in the projection $\mathcal{V}(\mathcal{M})$ because $\mathcal{V}$ does not satisfy their labels. In that case, it holds that $\mathcal{V}(\mathcal{M}) = \mathcal{V}(\mathcal{M}')$, although $\mathcal{M} \neq \mathcal{M}'$. The second reason is that a pinpointing rule application of a rule may change the projection (that is, $\mathcal{V}(\mathcal{M}) \neq \mathcal{V}(\mathcal{M}')$), but this change does not correspond to the application of the rule to $\mathcal{V}(\mathcal{M})$. For example, consider the rule $\mathsf{alc}_\exists$ and assume that we have an S-state containing the assertions $(\exists r.C)(a)$ with label ax_1 and $r(a, b), C(b)$ with label ax_2. Clearly, the rule $\mathsf{alc}_\exists$ is pinpointing applicable, and

its application adds the new assertions $r(a,c), C(c)$ both labeled with ax_1, where c is a new constant. Suppose now that $\mathcal{V}$ is a valuation that makes ax_1 and ax_2 true. Then the $\mathcal{V}$-projection of the S-state contains the three assertions $(\exists r.C)(a), r(a,b), C(b)$. Thus, the existential rule is not applicable, which means that no new individual c can be introduced. To overcome the second reason, we define a modified version of rule application in which the third condition for applicability from Definition 3.16 is removed.

Definition 3.25 (Modified rule application). *Given a S-state $\mathfrak{S} = (A, \mathcal{T})$, a rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$, and a substitution ρ on $\mathsf{var}(B_0)$, we say that R is* m-applicable to $\mathfrak{S}$ with ρ *if (i) $\mathcal{S} \subseteq \mathcal{T}$, and (ii) $B_0\rho \subseteq A$. In this case, we write $\mathcal{M} \to_{S^m} \mathcal{M}'$ if $\mathfrak{S} \in \mathcal{M}$ and $\mathcal{M}' = (\mathcal{M} \setminus \{\mathfrak{S}\}) \cup \{(A \cup B_i\sigma, \mathcal{T}) \mid 1 \leq i \leq m\}$, where σ is a substitution on the variables occurring in R that extends ρ and maps the fresh variables of R to distinct new constants.* ■

Modified rule applications are closely related to the "regular" rule applications as presented in Section 3.3 on one side, and to pinpointing rule applications on the other. In the following lemma, the term *saturated* refers to saturatedness with respect to $\to_S$, as introduced in Definition 3.16.

Lemma 3.26. *Let $\Gamma = (\mathcal{I}, \mathcal{T})$ be an axiomatised input and $\mathcal{M}_0 = \Gamma^S$.*

1. *Assume that $\mathcal{M}_0 \overset{*}{\to}_S \mathcal{M}$ and $\mathcal{M}_0 \overset{*}{\to}_{S^m} \mathcal{M}'$ where $\mathcal{M}$ and $\mathcal{M}'$ are saturated finite sets of S-states. Then $\mathcal{M}$ is full of clashes iff $\mathcal{M}'$ is full of clashes.*

2. *Assume that $\mathcal{M}$ and $\mathcal{M}'$ are finite sets of labeled S-states, and $\mathcal{V}$ a propositional valuation. Then $\mathcal{M} \to_{S^{pin}} \mathcal{M}'$ implies that either $\mathcal{V}(\mathcal{M}) \to_{S^m} \mathcal{V}(\mathcal{M}')$ or $\mathcal{V}(\mathcal{M}) = \mathcal{V}(\mathcal{M}')$. In particular, this shows that $\mathcal{M}_0 \overset{*}{\to}_{S^{pin}} \mathcal{M}$ implies $\mathcal{V}(\mathcal{M}_0) \overset{*}{\to}_{S^m} \mathcal{V}(\mathcal{M})$.*

Proof. The first statement of this lemma is a direct consequence of Lemma 3.31 that will be proved later in this section, and so we focus this proof only on the second statement.

Assume that $\mathcal{M} \to_{S^{pin}} \mathcal{M}'$; that is, there is an S-state $\mathfrak{S} = (A, \mathcal{T}) \in \mathcal{M}$ and a rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$ such that R is pinpointing applicable to $\mathfrak{S}$ with some substitution ρ and $\mathcal{M}' = (\mathcal{M} \setminus \{\mathfrak{S}\}) \cup \{(A \Cup_\psi B_i\sigma, \mathcal{T}) \mid 1 \leq i \leq m\}$ where σ and ψ are as in the definition of pinpointing application (Definition 3.20). Take an S-state $\mathfrak{S}_i = (A \Cup_\psi B_i\sigma, \mathcal{T}) \in \mathcal{M}'$ that was added by the application of R. By the definition of ψ-insertion, we know that (i) every assertion $a \in A \setminus \mathsf{ins}_\psi(B_i\sigma, A)$ keeps its old label $\mathsf{lab}(a)$, (ii) each newly added assertion in $\mathsf{ins}_\psi(B_i\sigma, A) \setminus A$ gets ψ as label, and (iii) every assertion $b \in A \cap \mathsf{ins}_\psi(B_i\sigma, A)$ modifies its label to $\psi \vee \mathsf{lab}(b)$. We will make a case analysis, depending on whether $\mathcal{V}$ satisfies the formula ψ or not.

If $\mathcal{V}$ satisfies ψ, then it holds that $(A \Cup_\psi B_i\sigma)_\mathcal{V} = A_\mathcal{V} \cup B_i\sigma$ since the label of each of the newly added assertions and each of the old assertions that got their label modified is implied by ψ and hence also satisfied by $\mathcal{V}$. This shows that $\mathcal{V}(\mathcal{M}) \to_{S^m} \mathcal{V}(\mathcal{M}')$ since the conditions of m-applicability follow directly from the fact that $\mathcal{V}$ satisfies ψ.

Consider now the case where $\mathcal{V}$ does not satisfy ψ. In this case we have that $(A \uplus_\psi B_i\sigma)_\mathcal{V} = A_\mathcal{V}$ since the label of every newly added assertion is ψ and hence not satisfied by $\mathcal{V}$, while the disjunction with ψ modifying the labels of the assertions in $A \cap B_i\sigma$ does not change the evaluation of the new labels under $\mathcal{V}$. It thus holds that $\mathcal{V}(\mathcal{M}) = \mathcal{V}(\mathcal{M}')$. □

If we have an axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$ and a sequence of rule applications $\mathcal{M}_0 \xrightarrow{*}_{S^{pin}} \mathcal{M}_n$ where $\mathcal{M}_0 = \Gamma^S$ and $\mathcal{M}_n$ is pinpointing saturated, we want to show that the clash formula $\psi = \psi_{\mathcal{M}_n}$ is in fact a pinpointing formula. This follows easily from the following two lemmas.

Lemma 3.27. *If $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$ then $\mathcal{V}$ satisfies ψ.*

Proof. Let $\mathcal{N}_0 = (\mathcal{I}, \mathcal{T}_\mathcal{V})^S$. Since S is a correct tableau, S must terminate on every input, and hence there exists a saturated set of S-states $\mathcal{N}$ such that $\mathcal{N}_0 \xrightarrow{*}_S \mathcal{N}$. By the same definition of correctness of S and the fact that $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$, we know that $\mathcal{N}$ is full of clashes. By Part 2 of Lemma 3.26, we know that $\mathcal{M}_0 \xrightarrow{*}_{S^{pin}} \mathcal{M}_n$ implies $\mathcal{V}(\mathcal{M}_0) \xrightarrow{*}_{S^m} \mathcal{V}(\mathcal{M}_n)$. Additionally, we know $\mathcal{V}(\mathcal{M}_0) = \mathcal{N}_0$, and by Lemma 3.23 that $\mathcal{V}(\mathcal{M}_n)$ is saturated. Thus, using 1 of Lemma 3.26 and the fact that $\mathcal{N}$ is full of clashes, we can deduce that $\mathcal{V}(\mathcal{M}_n)$ is also full of clashes. But then, by Lemma 3.22 we know that $\mathcal{V}$ satisfies $\psi = \psi_{\mathcal{M}_n}$. □

Lemma 3.28. *If $\mathcal{V}$ satisfies ψ then $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$.*

Proof. Consider as in the previous lemma a chain of rule applications of the form $\mathcal{N}_0 \xrightarrow{*}_S \mathcal{N}$ where $\mathcal{N}_0 = (\mathcal{I}, \mathcal{T}_\mathcal{V})^S$ and $\mathcal{N}$ is saturated. As S is a correct tableau for $\mathcal{P}$, in order to show that $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$, it suffices to prove that $\mathcal{N}$ is full of clashes. As in the proof of the previous lemma, we have that $\mathcal{V}(\mathcal{M}_0) \xrightarrow{*}_{S^m} \mathcal{V}(\mathcal{M}_n), \mathcal{V}(\mathcal{M}_0) = \mathcal{N}_0$, and $\mathcal{V}(\mathcal{M}_n)$ is saturated. Since $\mathcal{V}$ satisfies ψ, by Lemma 3.22 we know that $\mathcal{V}(\mathcal{M}_n)$ is full of clashes. By 1 of Lemma 3.26 this implies that $\mathcal{N}$ is also full of clashes. □

We have now completed the proof of Theorem 3.24, except for the first statement in Lemma 3.26. Before proving this result, we will introduce the notion of a *substate*. Intuitively, an S-state $\mathfrak{S}$ is a substate of an S-state $\mathfrak{S}'$ if every assertion and axiom in $\mathfrak{S}$ appears also in $\mathfrak{S}'$. However, we want to have a more general notion by allowing different constants to be used in the S-states as long as one can find a *renaming* of the constants in $\mathfrak{S}$ into the ones in $\mathfrak{S}'$ such that the desired inclusion between their sets of assertions holds.

Definition 3.29 (Substate). *The S-state $\mathfrak{S} = (A, \mathcal{T})$ is a* substate *of $\mathfrak{S}' = (A', \mathcal{T}')$, denoted as $\mathfrak{S} \preceq \mathfrak{S}'$ iff $\mathcal{T} \subseteq \mathcal{T}'$ and there is a* renaming function *$f : \mathsf{cons}(A) \to \mathsf{cons}(A')$ such that if $P(a_1, \ldots, a_k) \in A$, then $P(f(a_1), \ldots, f(a_k)) \in A'$.* ■

One important thing to notice is that if we have a pair of S-states $\mathfrak{S} = (A, \mathcal{T})$ and $\mathfrak{S}' = (A', \mathcal{T}')$ such that $\mathfrak{S} \preceq \mathfrak{S}'$, then the following property holds: if there is a set B of patterns and a substitution ρ on $\mathsf{var}(B)$ such that $B\rho \subseteq A$, then the substitution $\rho' = \rho \circ f$, where f is the renaming function that yields $\mathfrak{S} \preceq \mathfrak{S}'$, satisfies $B\rho' \subseteq A'$. In particular, this fact implies that $\mathfrak{S}'$ contains a clash whenever $\mathfrak{S}$ does.

Lemma 3.30. *Let $\mathcal{N}$ and $\mathcal{N}_0$ be sets of S-states, where $\mathcal{N}_0$ is saturated, and let $\mathfrak{S} \in \mathcal{N}$ and $\mathfrak{S}_0 \in \mathcal{N}_0$. If $\mathfrak{S} \preceq \mathfrak{S}_0$, then for every $\mathcal{N} \to_{\mathsf{R}^m} \mathcal{N}'$ there is $\mathfrak{S}' \in \mathcal{N}'$ such that $\mathfrak{S}' \preceq \mathfrak{S}_0$.*

Proof. If $\mathcal{N}'$ is obtained by the application of R to an S-state different from $\mathfrak{S}$ in $\mathcal{N}$, then $\mathfrak{S} \in \mathcal{N}'$ and thus nothing needs to be shown. Suppose then that the rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$ is applied to $\mathfrak{S}$ with some substitution ρ to obtain $\mathcal{N}'$, and let $\mathfrak{S} = (A, \mathcal{T})$ and $\mathfrak{S}_0 = (A_0, \mathcal{T}_0)$. Since $\mathfrak{S} \preceq \mathfrak{S}_0$, it holds that $\mathcal{S} \subseteq \mathcal{T} \subseteq \mathcal{T}_0$ and that there is a substitution ρ' on $\mathsf{var}(B_0)$ such that $B_0\rho' \subseteq A_0$. This all means that conditions (i) and (ii) from the definition of rule applicability are satisfied for $\mathfrak{S}_0$, R and ρ'. Since $\mathcal{N}_0$ is saturated, R cannot be applicable to $\mathfrak{S}_0$ with ρ', and hence condition (iii) cannot hold. This means that there must exist an $i, 1 \leq i \leq m$ and a substitution ϱ on $\mathsf{var}(B_0 \cup B_i)$ extending ρ' such that $B_i\varrho \subseteq A_0$.

On the other hand, a substitution σ extending ρ was used to construct the new set $\mathcal{N}'$ of S-states through the application of the rule R to $\mathfrak{S}$. Let $\mathfrak{S}' = (A \cup B_i\sigma, \mathcal{T})$. Since σ maps the fresh variables of R to distinct new constants, we can extend the renaming function f to $f' : \mathsf{cons}(A \cup B_i\sigma) \to \mathsf{cons}(A_0)$ by setting $f'(\sigma(x)) = \varrho(x)$ for every fresh variable x of R appearing in B_i. This defines a complete renaming function f' for the constants in $A \cup B_i\sigma$ and by definition this function satisfies $\sigma \circ f' = \varrho$.

We show now that $\mathfrak{S}' \preceq \mathfrak{S}_0$ by means of the new renaming function f'. Let $P(a_1, \ldots, a_k) \in A \cup B_i\sigma$. If this assertion belongs to A, then, since $\mathfrak{S} \preceq \mathfrak{S}_0$ with the renaming function f, it holds that $P(f'(a_1), \ldots, f'(a_k)) = P(f(a_1), \ldots, f(a_k)) \in A_0$. If $P(a_1, \ldots, a_k) \in B_i\sigma$, then $P(a_1, \ldots, a_k) = P(\sigma(x_1), \ldots, \sigma(x_k))$ for some variables $x_1, \ldots, x_k \in \mathsf{var}(B_0 \cup B_i)$. But since $\sigma \circ f' = \varrho$, we have

$$P(f'(a_1), \ldots, f'(a_k)) = P(\varrho(x_1), \ldots, \varrho(x_k)) \in B_i\varrho \subseteq A_0,$$

which completes the proof that $\mathfrak{S}' \preceq \mathfrak{S}_0$. □

The following lemma generalises the first part of Lemma 3.26.

Lemma 3.31. *Let Γ be an axiomatised input and $\mathcal{M}_0 = \Gamma^S$. If $\mathcal{M}$ and $\mathcal{M}'$ are saturated sets of S-states such that $\mathcal{M}_0 \xrightarrow{*}_{S^m} \mathcal{M}$ and $\mathcal{M}_0 \xrightarrow{*}_{S^m} \mathcal{M}'$, then $\mathcal{M}$ is full of clashes iff $\mathcal{M}'$ is full of clashes.*

Proof. Recall that the application of a rule to a set of S-states removes one of these S-states and adds a finite number of S-states that extend the removed one. Thus, for every S-state $\mathfrak{S} \in \mathcal{M}'$ there is an S-state $\mathfrak{S}_0 \in \mathcal{M}_0$ such that $\mathfrak{S}_0 \preceq \mathfrak{S}$.

Consider the chain of (modified) rule applications

$$\mathcal{M}_0 \to_{S^m} \mathcal{M}_1 \to_{S^m} \ldots \to_{S^m} \mathcal{M}_n = \mathcal{M}$$

that leads from $\mathcal{M}_0$ to $\mathcal{M}$. Since $\mathcal{M}'$ is saturated, we can use Lemma 3.30 to deduce that for every $\mathfrak{S} \in \mathcal{M}'$ there is an S-state $\mathfrak{S}_1 \in \mathcal{M}_1$ such that $\mathfrak{S}_1 \preceq \mathfrak{S}$. By iterating this argument, we obtain that, for every $\mathfrak{S} \in \mathcal{M}'$ there is an element $\mathfrak{S}_n \in \mathcal{M}$ such that $\mathfrak{S}_n \preceq \mathfrak{S}$.

Assume now that $\mathcal{M}$ is full of clashes; that is, every S-state in $\mathcal{M}$ contains a clash and take an arbitrary $\mathfrak{S} \in \mathcal{M}'$. We must show that $\mathfrak{S}$ contains a clash. As shown in

the previous paragraph, there is an element $\mathfrak{S}_n \in \mathcal{M}$ such that $\mathfrak{S}_n \preceq \mathfrak{S}$. The fact that $\mathfrak{S}_n$ contains a clash implies that $\mathfrak{S}$ contains also a clash. This finishes the proof of the *only if* direction. A symmetric argument can be used to prove the converse direction. □

When proving the correctness of the pinpointing extension of a tableau, we consider only terminating chains of pinpointing rule applications. Unfortunately, although a correct tableau needs to be terminating, this property not necessarily transfer to its pinpointing extension. The reason for this is that a rule may be pinpointing applicable in cases where it is not applicable in the normal sense, as discussed before. Even if we restrict ourselves to deterministic rules, the problem still appears, as shown in the following example.

Example 3.32. *Consider the tableau S with the following three rules*[13]

$$\begin{aligned}
\mathsf{R}_1 &: (\{P(x)\}, \{ax_1\}) \to \{r(y,y,y), Q_1(y), Q_2(y)\},\\
\mathsf{R}_2 &: (\{P(x)\}, \{ax_2\}) \to \{r(y,y,y), Q_1(y), Q_2(y)\},\\
\mathsf{R}_3 &: (\{Q_1(x), Q_2(y)\}, \emptyset) \to \{r(x,y,z), Q_1(y), Q_2(z)\},
\end{aligned}$$

where the function $\cdot^S$ maps every input $\mathcal{I} \in \mathfrak{I}$ to the set $\{P(a)\}$ and every axiom from $\mathfrak{T} = \{ax_1, ax_2\}$ to the empty set, with $\mathscr{P}_{admis}(\mathfrak{T}) = \mathscr{P}(\mathfrak{T})$. For any axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$, we have $\Gamma^S = (\{P(a)\}, \mathcal{T})$. Depending on the axioms appearing in $\mathcal{T}$, the rules R_1 and/or R_2 may be applicable to this S-state, but R_3 is not. Notice that R_1 and R_2 have the same right-hand side, and thus the application of any of them to Γ^S leads to the same S-state modulo the chosen new constant name introduced for the fresh variable y. Suppose we apply one of these rules and introduce the new constant b. The resulting S-state is $\mathfrak{S} = (A, \mathcal{T})$ where

$$A = \{P(a), Q_1(b), Q_2(b), r(b,b,b)\}.$$

No rule is then applicable to $\mathfrak{S}$. In fact, in order to apply any of the rules $\mathsf{R}_1, \mathsf{R}_2$, the only way to satisfy Condition (ii) from the definition of rule application (Definition 3.6) is to use a substitution that maps the variable x to the constant a. By extending this substitution to map y to the constant b, Condition (iii) from the same definition is violated since the assertions $Q_1(b), Q_2(b)$ and $r(b,b,b)$ already appear in $\mathfrak{S}$, after being introduced by the first rule application. To satisfy Condition (ii) for rule R_3, we must choose the substitution ρ that maps both variables x and y to the constant b. If we extend ρ to map z to the same constant b we then violate Condition (iii). This all shows that S indeed terminates on every axiomatised input; in fact, at most one rule is applicable before reaching a saturated S-state.

However, it is possible to construct an infinite chain of pinpointing rule applications starting with $\Gamma^S = (\{P(a)\}, \{ax_1, ax_2\})$, where $\mathsf{lab}(P(a)) = \top$. We can first apply rule R_1 to obtain the S-state $\mathfrak{S}$ described above, where all the assertion, with the exception of $P(a)$, are labeled with ax_1. At this point, rule R_2 is pinpointing applicable since,

[13] Since all the rules are deterministic and hence there will always be only one S-state, we express only this state, instead of the set containing it.

although there is an extension of the substitution under which all the assertions exist already in $\mathfrak{S}$, these assertions are labeled with the formula ax_1, which is not implied by ax_2. The pinpointing application of R_2 to $\mathfrak{S}$ adds the assertions $Q_1(c), Q_2(c)$ and $r(c,c,c)$ all with label ax_2. It is now possible to apply the rule R_3 to the resulting S-state $\mathfrak{S}'$ with the substitution ρ mapping the variables x and y to the constants b and c, respectively. Since the S-state $\mathfrak{S}'$ does not contain any assertion of the form $r(b,c,\_)$, Condition (iii) cannot be violated. This rule application adds the assertions $r(b,c,d), Q_2(d)$ with label $ax_1 \wedge ax_2$. It is easy to see that the rule R_3 can be now repeatedly applied, producing this way a non-terminating chain of pinpointing rule applications. ■

This example shows that the termination of a tableau S does not necessarily imply the termination of its pinpointing extension, even for the restricted case of tableaux having only deterministic rules. In Chapter 6 we will show that it is in general undecidable whether the pinpointing extension of a tableau is terminating. Nonetheless, we can still search for classes of tableaux that have terminating extensions. Moreover, as shown in Sections 2.3.4 and 2.3.5, some tableau algorithms actually require additional techniques to ensure termination, and those techniques need to be adapted to pinpointing extensions as well in order to preserve correctness. The next chapter deals with termination of pinpointing extensions in both fronts. First it introduces a class of terminating tableaux whose pinpointing extensions are always terminating. Afterwards, it defines a general notion of blocking, taking as model the notion of equality blocking from Section 2.3.5, and shows how it can be extended to produce a correct and terminating pinpointing procedure.

Chapter 4

A Class of Terminating Tableaux

The pinpointing extension of general tableaux presented in the previous chapter requires a relaxation of the rule-applicability conditions to ensure that all possible ways in which a property can be deduced are detected in a single execution. Example 3.32 shows that these relaxed applicability conditions may lead to a non-terminating procedure. This undesired behaviour may arise even in restricted scenarios, as when only deterministic rules are allowed. Since we are interested in describing a terminating procedure, we turn our attention to the causes of termination of known tableau algorithms, aiming towards a framework that not only ensures the termination of the original tableau algorithms, but also transfers this result to their pinpointing extensions.

We identify tableaux that generate tree-like S-states as good candidates for termination. On one side, if we are able to bound the breadth and depth of these S-states, there will be no way an infinite chain of rule applications can be generated. On the other, even if we are unable to bound the depth of these trees, we can reuse the ideas of blocking to avoid generating an infinite tree. The tree-like structure is important for blocking for two reasons: first, we need a notion of *nodes* to have one blocking another, and second, the tree shape yields a natural ordering that allows us to forbid mutual blocking by two nodes, which would lead to an incorrect procedure. Actually, we allow for a slightly more general scenario, in what we will call *forest tableaux*. These tableaux, which are formally defined in Section 4.1, may produce several trees that "grow" from an arbitrary graph-like structure. Using this notion, we first present additional conditions that bound the growth of the trees generated by these tableaux, and show that they suffice for ensuring termination in Section 4.2. Finally, in Section 4.3, we introduce a general notion of blocking analogous to equivalence blocking introduced in Chapter 2, and show how it can be used to ensure an answer in finite time.

4.1 Forest Tableaux

One of the reasons why tableau algorithms for certain DLs terminate is that they create a tree-like structure for which the out-degree and the depth of the tree are bounded

by a function of the size of the input formula. The nodes of these trees are labeled, but the input determines a finite number of possible labels. A typical example is the tableau-based decision procedure for satisfiability of $\mathcal{ALC}$-concepts (see Chapter 2). This algorithm generates sets of assertions of the form $r(a, b)$ and $C(a)$, where r is a role and C is an $\mathcal{ALC}$-concept description. The tree structure is induced by role assertions, and the nodes are labeled by sets of concepts, i.e., node a is labeled with $\{C_1, \ldots, C_n\}$ if $C_1(a), \ldots, C_n(a)$ are all the concept assertions involving a. The main reasons why the algorithm terminates are:

- the depth of the tree structure is bounded by the size n of the input, i.e., the maximal length m of chains $r_1(a_0, a_1), r_2(a_1, a_2), \ldots, r_m(a_{m-1}, a_m)$ in a set of assertions generated by the algorithm is bounded by n;
- the out-degree of the tree structure is bounded by n , i.e., the maximal number m of assertions $r_1(a_0, a_1), r_2(a_0, a_2), \ldots, r_m(a_0, a_m)$ in a set of assertions generated by the algorithm is bounded by n;
- for every assertion $C(a)$ occurring in a set of assertions generated by the algorithm, C is a sub-description of the input concept description.

If we look at the algorithm that decides consistency of $\mathcal{ALC}$-ABoxes (Section 2.3.2) then things are a bit more complicated: rather than a single tree one obtains a forest, more precisely, several trees growing out of the input ABox. But these trees satisfy the restrictions mentioned above, which is enough to show termination.

Basically, we want to formalise this reason for termination within the general framework of tableaux introduced in the previous chapter. However, to be as general as possible, we do not want to restrict assertions to be built from unary predicates (concepts) and binary predicates (roles) only. For this reason, we allow for predicates of arbitrary arity, but restrict our assertions such that states (i.e., sets of assertions) induce graph-like structures. This general approach allows us to model, among others, the tableaux decision algorithm for the n-ary DL $\mathcal{GF}1^-$ introduced in [LST99].

In order to have a graph-like structure, we must be able to distinguish between nodes and edges. For this reason, we now assume that the signature Σ is partitioned into the sets Λ and Δ, where each predicate name $P \in \Lambda$ is equipped with an arity n, while every predicate name $r \in \Delta$ is equipped with a double arity $0 < m < n$. Strictly speaking, the arity of $r \in \Delta$ is n; however, the first m argument positions are grouped together, as are the last $n - m$. Intuitively, the elements of Λ correspond to DL concepts and form the nodes of the graph-like structure, whereas the elements of Δ correspond to DL roles and induce the edges.

If a pattern/assertion p starts with a predicate from Δ (Λ), we say that p is a *Δ-pattern/assertion (Λ-pattern/assertion)*, and write $p \in \widehat{\Delta}$ ($p \in \widehat{\Lambda}$). In our $\mathcal{ALC}$ example, the set Λ consists of all $\mathcal{ALC}$-concepts, which have arity 1, and Δ consists of all role names, which have double arity $1, 2$. For the rest of this chapter, assertions and patterns in $\widehat{\Lambda}$ will be denoted using capital letters ($P, Q, R, \ldots$), and those in $\widehat{\Delta}$ using lower-case letters ($r, s, t, \ldots$). Given a predicate $p \in \Delta$ with double arity m, n, the sets of *parents* and *descendants* of the pattern $r = p(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n)$ are given by $\overleftarrow{r} = \{x_1, \ldots, x_m\}$ and $\overrightarrow{r} = \{x_{m+1}, \ldots, x_n\}$, respectively.

In the different tableau algorithms presented in Chapter 2 for deciding properties in $\mathcal{ALC}$, the nodes of the trees are defined by the constants occurring in the set of assertions, and the concept assertions give rise to the labels of these nodes. In the general case, nodes are not single constants, but rather sets of assertions built over a *connected* set of constants.

Definition 4.1 (Connected). *Let B be a set of Σ-patterns (Σ-assertions), and $x, y \in \mathsf{var}(B)$ ($a, b \in \mathsf{cons}(B)$). We say that x and y (a and b) are* B-connected, *denoted as $x \sim_B y$ ($a \sim_B b$), if there are variables $x_0, x_1, \ldots, x_n \in \mathsf{var}(B)$ (constants $a_0, a_1, \ldots, a_n \in \mathsf{cons}(B)$) and patterns $P_1, \ldots, P_n \in B \cap \widehat{\Lambda}$ (respectively assertions $P_1, \ldots, P_n \in B \cap \widehat{\Lambda}$) such that $x = x_0, y = x_n$ ($a = a_0, b = a_n$) and for every $1 \leq i \leq n$ it holds that $\{x_{i-1}, x_i\} \subseteq \mathsf{var}(P_i)$ ($\{a_{i-1}, a_i\} \subseteq \mathsf{cons}(P_i)$).*

We say that B is connected *if, for every $x, y \in \mathsf{var}(B)$ ($a, b \in \mathsf{cons}(B)$), we have $x \sim_B y$ ($a \sim_B b$).* ■

Connected sets of assertions can be viewed as *bundles* that join the constants contained in them. Nodes will be formed by maximal connected sets of assertions from $\widehat{\Lambda}$. An assertion from $\widehat{\Delta}$ will be treated as a (directed) edge that connects a node containing its parent constants with a node containing its descendant constants.

Definition 4.2 (Graph structure). *Let B be a set of assertions. A maximal connected subset $N \subseteq B \cap \widehat{\Lambda}$ is called a* node *in B. An assertion $r \in B \cap \widehat{\Delta}$ is called an* edge *in B if there are two nodes N_1 and N_2 in B such that $\overleftarrow{r} \subseteq \mathsf{cons}(N_1)$ and $\mathsf{cons}(N_2) \subseteq \overrightarrow{r}$. In this case, we say that r* connects *N_1 to N_2.*

The set B is a graph structure *if every $r \in B \cap \widehat{\Delta}$ is an edge. If B is a graph structure, the corresponding B-*graph *$\mathcal{G}_B$ contains one vertex v_N for every node N, and an edge (v_N, v_M) if there is an edge connecting N to M.*

The notion of a graph structure and of the corresponding graph can be extended to states $\mathfrak{S} = (B, \mathcal{T})$ in the obvious way: $\mathfrak{S}$ is a graph structure *if B is one, and in this case $\mathcal{G}_{\mathfrak{S}} := \mathcal{G}_B$.* ■

If a set of assertions B is a graph structure, then the set of nodes forms a partition of $B \cap \widehat{\Lambda}$, and each of its elements either belongs to a node or is a (directed) edge. Observe, however, that an edge $r \in \widehat{\Delta}$ may connect a node with more than one successor node. For example, consider the set of assertions $B = \{P(a), Q(b), R(c), r(a, b, c)\}$ where $P, Q, R \in \Lambda$ are unary, and $r \in \Delta$ has double arity $1, 3$. This set forms a graph structure consisting of the nodes $N_1 := P(a), N_2 := Q(b), N_3 := R(c)$ and the edge $r(a, b, c)$. This single edge connects N_1 to both N_2 and N_3. $\mathcal{G}_B$ is then the graph $(\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_1, v_3)\})$. This will create no problem in our proofs, but must be kept in mind when dealing with graph-structures and their corresponding graphs.

Recall that the tableau-based decision procedure for consistency of $\mathcal{ALC}$-ABoxes (Section 2.3.2) starts with an ABox, which can be viewed as a graph, but then extends this ABox by trees that grow out of the nodes of this graph. The following definition introduces *forest tableaux*, which show a similar behavior, but are based on the more general notion of a graph structure introduced above.

Definition 4.3 (Forest tableau). *The tableau* $S = (\Sigma, \cdot^S, \mathcal{R}, \mathcal{C})$ *is called a* forest tableau *if for every axiomatised input* Γ *and every* $\mathfrak{S} \in \Gamma^S$*, the state* $\mathfrak{S}$ *is a graph structure, every clash* $C \in \mathcal{C}$ *is a connected subset of* $\widehat{\Lambda}$*, and the following conditions hold for every rule* $(B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$ *and every* $1 \leq i \leq m$*:*

1. *for every* Σ*-pattern* $r \in B_0 \cap \widehat{\Delta}$*, there exists a* Σ*-pattern* $P \in B_0 \cap \widehat{\Lambda}$ *such that* $\overleftarrow{r} \subseteq \mathsf{var}(P)$ *or* $\overrightarrow{r} \subseteq \mathsf{var}(P)$*.*
2. *for every* Σ*-pattern* $r \in B_i \cap \widehat{\Delta}$*, there exists a* Σ*-pattern* $P \in B_0 \cap \widehat{\Lambda}$ *such that* $\overleftarrow{r} \subseteq \mathsf{var}(P)$*.*
3. *for every* Σ*-pattern* $r \in B_i \cap \widehat{\Delta}$*, we have* $\overrightarrow{r} \cap \mathsf{var}(B_0) = \emptyset$*.*
4. *if* $r, s \in B_i \cap \widehat{\Delta}$ *are distinct patterns, then* $\overrightarrow{r} \cap \overrightarrow{s} = \emptyset$*.*
5. *for every* Σ*-pattern* $P \in B_i \cap \widehat{\Lambda}$*, either*
   (i) there is a Σ*-pattern* $r \in (B_0 \cup B_i) \cap \widehat{\Delta}$ *such that* $\mathsf{var}(P) \subseteq \overrightarrow{r}$ *or* $\mathsf{var}(P) \subseteq \overleftarrow{r}$*, or*
   (ii) there is a $Q \in B_0 \cap \widehat{\Lambda}$ *with* $\mathsf{var}(P) \subseteq \mathsf{var}(Q)$*.*
6. *if* $B_0 \cap \widehat{\Delta} \neq \emptyset$*, then* $B_i \cap \widehat{\Delta} = \emptyset$*.*
7. $B_0 \cap \widehat{\Lambda}$ *is connected.*

■

A few intuitive explanations for these conditions are in order. *Condition 1* ensures that every edge triggering a rule application is connected to a node, which may be either a parent or a descendant node of this edge. *Condition 2* makes sure that for every newly introduced edge, a parent node was present before the rule is applied. This implies that a rule application cannot add new predecessors to a node, and that newly introduced nodes are not disconnected from the rest of the graph structure. Both of these properties are vital for obtaining forest-like structures. *Condition 3* states that every newly generated edge has only new constants in its descendant set. In other words, new edges cannot connect old nodes, but only generate new nodes as descendant. *Condition 4* ensures that, even if several edges are added by a single rule application, these edges connect different nodes with the parent node, avoiding this way that a node is connected by multiple edges to a parent node. *Condition 5* makes sure that we always have a connected graph. It states that, whenever a non-edge assertion is added, it must either belong to an old node, or belong to a descendant node added by the creation of a new edge within the same rule application. *Condition 6* states that the addition of new edges must only depend on the assertions belonging to the parent nodes, but never on the presence of other edges. In particular, this ensures that each descendant is created independently from its siblings, as long this is done in distinct rule applications. Finally, *Condition 7* ensures that the non-edge assertions triggering a rule application all belong to the same node.

The different (disjunctive) options stated in Conditions 1 and 5(i) require an additional explanation. They allow the tableau rules to propagate information not just

to successor nodes, but also to predecessor nodes in the trees. The main reason for including this possibility in our framework is that it makes it general enough to deal with constructors such as *inverse roles* in DLs, and hence model $\mathcal{SI}$-TBoxes. The price to pay for this decision is twofold: on the one side, more cases must be analysed in the proofs. On the other, the weaker version of blocking, *subset blocking*, will not suffice to yield a correct terminating algorithm (see Example 2.12) and we will have to use an analogous to *equivalence blocking*. Notice nonetheless that if the use of subset blocking leads to a correct decision procedure, using instead equivalence blocking will still yield a correct answer, though its efficiency may be compromised as the cycles will take longer to be detected.

Although this definition may seem to complex at first sight, all the conditions are local for each rule and only impose restrictions on their syntactic form; thus, they can be easily verified to determine whether a given tableau belongs to the class of forest tableaux or not.

The following lemma shows that the S-states of a forest tableau form graph structures in which every node is connected to an initial node via a series of edges. We show that it is actually the case even for modified rule applications, since we want to use it also for the pinpointing extensions. Its proof is identical to that of Lemma 4.7, by simply deleting every reference to the ordering relation used there. To avoid a futile repetition of the lengthy proof, we do not present this proof here, but delay it to the following section.

Lemma 4.4. *Let S be a forest tableau, Γ an axiomatised input, $\mathfrak{S}_0 \rightarrow_{S^m} \mathfrak{S}_1 \rightarrow_{S^m} \cdots$ a sequence of modified rule applications, and $\mathfrak{S}_0 \in \Gamma^S$. Then, for every $\mathfrak{S}_i = (A_i, \mathcal{T})$ and $P \in A_i \cap \widehat{\Lambda}$, either $\mathsf{cons}(P) \subseteq \mathsf{cons}(A_0)$ or there are $r \in A_i \cap \widehat{\Delta}$ and $Q \in A_i \cap \widehat{\Lambda}$ such that $\overleftarrow{r} \subseteq \mathsf{cons}(Q)$, and $\mathsf{cons}(P) \subseteq \overrightarrow{r}$.*

In fact, due to Conditions 3 and 4 of Definition 4.3, we can deduce that the r described by this lemma is unique for every given P. Thus, the S-states of a forest tableau form indeed a forest structure as described before.

Clearly, just ensuring that all states generated by a tableau have a forest-like structure is not sufficient to yield termination. We must also ensure that the trees in the forest cannot grow indefinitely (i.e., that the overall number of nodes that can be generated is bounded), and that the same is true for the nodes (i.e., that the number of assertions making up a single node is bounded). To bound the number of possible assertions, we restrict the set of predicate names that can be used; this restricted set is called a *cover*.

Definition 4.5 (Cover). *Let $S = (\Sigma, \cdot^S, \mathcal{R}, \mathcal{C})$ be a tableau and $\mathcal{T}$ a set of axioms. A set $\Omega \subseteq \Sigma$ is called a* $\mathcal{T}$-cover *if, for every rule $\mathsf{R} : (B_0, \mathcal{S}) \rightarrow \{B_1, \ldots, B_n\}$ such that $\mathcal{S} \subseteq \mathcal{T}$ and B_0 contains only predicates from Ω, the sets B_i for $i = 1, ..., n$ also contain only predicates from Ω.*

The tableau S is covered *if, for every axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$, there is a finite $\mathcal{T}$-cover Ω_Γ such that every S-state in Γ^S contains only predicates from Ω_Γ.* ■

Given such a covered tableau, every state that can be reached from an initial state in Γ^S by applying rules from S contains only predicates from Ω_Γ. We will see that

this ensures that nodes cannot grow indefinitely.

To prevent the trees from growing indefinitely (i.e., to bound the number of nodes), it is enough to enforce finite branching and finite paths in the trees. Finite branching actually already follows from the conditions we have stated so far. Hence, we need only to make sure that paths cannot get indefinitely long. The next section shows how a partial order can be used to ensure this.

4.2 Ordered Tableaux

To bound the length of paths, we additionally require the predicates occurring in rules to be decreasing w.r.t. a given partial order, in such a way that nodes farther away from the root will have smaller predicates than their predecessors. Given a strict partial order $<$ on predicates, we extend it to patterns (assertions) by defining $P < Q$ if the predicate of the pattern (assertion) P is smaller than the predicate of the pattern (assertion) Q.

Definition 4.6 (Ordered tableaux). *A covered tableau S is called an* ordered tableau *if, for every axiomatised input Γ, there is a strict partial ordering $<_\Gamma$ on the predicate names in $\Omega_\Gamma \cap \Lambda$ such that, for every rule $(B_0, \mathcal{S}) \to \{B_1, \ldots, B_n\}$, every $1 \leq i \leq n$, and every $P \in B_0 \cap \widehat{\Lambda}$ and $Q \in B_i \cap \widehat{\Lambda}$, we have $Q <_\Gamma P$.* ■

For example, the tableau-based decision procedure for consistency of $\mathcal{ALC}$-ABoxes is an ordered tableau. It is covered since rule application only adds concept assertions $C(a)$ (role assertions $r(a,b)$) where C is a sub-description of a concept description occurring in the input ABox $\mathcal{A}_0$ (where r is a role occurring in the input ABox $\mathcal{A}_0$). Thus one can take the set of sub-descriptions of concept descriptions occurring in $\mathcal{A}_0$ together with the roles occurring in $\mathcal{A}_0$ as a cover. In addition, rule application only adds concept assertions that either have a smaller role-depth (i.e., nesting of existential and value-restrictions) than the one that triggered it, or are subconcepts of it. Thus, ordering concept descriptions by their role-depth and by the subconcept relation yields the desired partial order.

Ordered tableaux have the property that, if applied to an axiomatised input Γ, none of the trees in the generated forest can have a depth greater than the cardinality of the cover Ω_Γ. This easily follows from the next lemma.

Lemma 4.7. *Let S be an ordered forest tableau, Γ an axiomatised input, $\mathfrak{S}_0 \in \Gamma^S$, and $\mathfrak{S}_0 \to_{S^m} \mathfrak{S}_1 \to_{S^m} \cdots$ a sequence of modified rule applications. Then, for every $\mathfrak{S}_i = (A_i, \mathcal{T})$ and $P \in A_i \cap \widehat{\Lambda}$, either $\mathsf{cons}(P) \subseteq \mathsf{cons}(A_0)$ or there are $r \in A_i \cap \widehat{\Delta}$ and $Q \in A_i \cap \widehat{\Lambda}$ such that $\overleftarrow{r} \subseteq \mathsf{cons}(Q)$, $\mathsf{cons}(P) \subseteq \overrightarrow{r}$, and $P <_\Gamma Q$.*

Proof. The proof is by induction on i. For $\mathfrak{S}_0$ the result is trivial. Suppose now that it holds for $\mathfrak{S}_i$, and that the rule $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_n\}$ is applied to $\mathfrak{S}_i$ to obtain $\mathfrak{S}_{i+1} = (A_{i+1}, \mathcal{T})$, where $A_{i+1} = A_i \cup B_j\sigma$ for some substitution σ and some $j, 1 \leq j \leq n$. Let $P \in A_{i+1} \cap \widehat{\Lambda}$. If $P \in A_i$, then by the induction hypothesis and the fact that $A_i \subseteq A_{i+1}$, the result holds. Otherwise, P was added by the application of

R. By Condition 5 of Definition 4.3, we have either (i) an $r \in (B_0 \cup B_j)\sigma \cap \widehat{\Delta}$ with $\mathsf{cons}(P) \subseteq \overrightarrow{r}$ or $\mathsf{cons}(P) \subseteq \overleftarrow{r}$, or (ii) there is a $Q \in B_0\sigma \cap \widehat{\Lambda}$ with $\mathsf{cons}(P) \subseteq \mathsf{cons}(Q)$.

We will analyse Case (ii) first. Since the rule was applied with substitution σ, we have $B_0\sigma \subseteq A_i$, and thus $Q \in A_i \cap \widehat{\Lambda}$. Since S is ordered, we also know that $P <_\Gamma Q$. By the induction hypothesis, either $\mathsf{cons}(Q) \subseteq \mathsf{cons}(A_0)$, or $\overleftarrow{r} \subseteq \mathsf{cons}(Q')$, $\mathsf{cons}(Q) \subseteq \overrightarrow{r}$, and $Q <_\Gamma Q'$ for assertions $r, Q' \in A_i$. In both cases, transitivity of $<_\Gamma$ and of $\subseteq$ yield the desired result.

We focus now on Case (i). Suppose first that $\mathsf{cons}(P) \subseteq \overrightarrow{r}$. If $r \in B_j\sigma$, then by Condition 2 of Definition 4.3, there is a $Q \in B_0\sigma \subseteq A_i$ such that $\overleftarrow{r} \subseteq \mathsf{cons}(Q)$. Since S is ordered, we also have $P <_\Gamma Q$, which completes the proof for the case where $\mathsf{cons}(P) \subseteq \overrightarrow{r}$ and $r \in B_j\sigma$.

Next, we consider the case where $\mathsf{cons}(P) \subseteq \overrightarrow{r}$ and $r \in B_0\sigma$. Then, by Condition 1 of Definition 4.3, there must exist a $Q \in B_0\sigma$ such that $\overleftarrow{r} \subseteq \mathsf{cons}(Q)$ or $\overrightarrow{r} \subseteq \mathsf{cons}(Q)$. In the former case, the proof is analogous to the one for the first part of this case. In the latter case, we have $\mathsf{cons}(P) \subseteq \overrightarrow{r} \subseteq \mathsf{cons}(Q)$, which is an instance of Case (ii).

Finally, suppose that $\mathsf{cons}(P) \subseteq \overleftarrow{r}$. We can assume without loss of generality that there is no $Q \in B_0\sigma \cap \widehat{\Lambda}$ such that $\mathsf{cons}(P) \subseteq \mathsf{cons}(Q)$. In fact, if it existed, we would be in Case (ii) analysed above. Consequently, r cannot belong to $B_i\sigma$ since this would violate Condition 2 of Definition 4.3. Hence, $r \in B_0\sigma$ and there must exist a $Q \in B_0\sigma \cap \widehat{\Lambda}$ such that $\overleftarrow{r} \subseteq \mathsf{cons}(Q)$ or $\overrightarrow{r} \subseteq \mathsf{cons}(Q)$.

In the first case, we have $\mathsf{cons}(P) \subseteq \overleftarrow{r} \subseteq \mathsf{cons}(Q)$, which brings us back to Case (ii) analysed above. In the other case, we know that $P <_\Gamma Q$ and $Q \in A_i$. Thus, by the induction hypothesis, the statement of the lemma holds for Q.

If $\mathsf{cons}(Q) \subseteq \mathsf{cons}(A_0)$, then—due to our assumption in this case stating that $\overrightarrow{r} \subseteq \mathsf{cons}(Q)$—we also have $\overrightarrow{r} \subseteq \mathsf{cons}(A_0)$. This means that r was not added by any previous rule application as otherwise this would violate Condition 3 of Definition 4.3. Thus, r must have been already present in A_0, which implies $\overleftarrow{r} \subseteq \mathsf{cons}(A_0)$. Since $\mathsf{cons}(P) \subseteq \overleftarrow{r}$, it also holds that $\mathsf{cons}(P) \subseteq \mathsf{cons}(A_0)$.

Now, assume that $\mathsf{cons}(Q) \not\subseteq \mathsf{cons}(A_0)$. By the induction hypothesis, there exist $s \in A_i \cap \widehat{\Delta}$ and $R \in A_i \cap \widehat{\Lambda}$ such that $\overleftarrow{s} \subseteq \mathsf{cons}(R), \mathsf{cons}(Q) \subseteq \overrightarrow{s}$, and $Q <_\Gamma R$. Since $\mathsf{cons}(Q) \not\subseteq \mathsf{cons}(A_0)$, we know that Q and s were added by a (previous) rule application. We claim that $r = s$. In fact, we have $\emptyset \neq \overrightarrow{r} \subseteq \mathsf{cons}(Q) \subseteq \overrightarrow{s}$. If we had $r \neq s$, then this would violate Condition 3 or 4 of Definition 4.3, where Condition 3 covers the case where r and s are introduced by different rule applications, and Condition 4 covers the case where these two assertions are added by the same rule application.

Overall, we thus know that $\mathsf{cons}(P) \subseteq \overleftarrow{r} \subseteq \mathsf{cons}(R)$ and $P <_\Gamma R$. Since $R \in A_i$, by the induction hypothesis, we have once again that either $\mathsf{cons}(R) \subseteq \mathsf{cons}(A_0)$ or there exist $r' \in A_i \cap \widehat{\Delta}$ and $Q' \in A_i \cap \widehat{\Lambda}$ such that $\overleftarrow{r'} \subseteq \mathsf{cons}(Q'), \mathsf{cons}(R) \subseteq \overrightarrow{r'}$, and $R <_\Gamma Q'$. In both cases, the fact that $\mathsf{cons}(P) \subseteq \mathsf{cons}(R)$ and $P <_\Gamma R$, together with the transitivity of $\subseteq$ and $<_\Gamma$, yields the desired result. □

Notice that in this proof, the existence of the stated assertions r and Q does not depend on the fact that the tableau is ordered, or even covered. Those restrictions

are only used for showing that indeed there is a decreasing sequence of predicates in each $\mathfrak{S}_i$. Hence, removing all references to this ordering yields a proof for Lemma 4.4.

An easy consequence of Lemma 4.7 is that a path consisting of m new edges in a state generated by rule applications from a state in Γ^S implies a decreasing sequence w.r.t. $<_\Gamma$ of the same length. Consequently, the length of such paths is bounded by the number of predicate symbols occurring in the finite cover Ω_Γ.

Proposition 4.8. *Let $\mathfrak{S}_0 \stackrel{*}{\rightarrow}_{S^m} \mathfrak{S}$ where $\mathfrak{S}_0 = (A_0, \mathcal{T}) \in \Gamma^S$ and $\mathfrak{S} = (A, \mathcal{T})$. Suppose that A contains edges $r_1, \ldots, r_m$ and nodes $N_0, \ldots, N_m$ such that for all $i, 1 \leq i \leq m$, $r_i \notin A_0$ and r_i connects N_{i-1} with N_i. Then, there exist assertions $Q_1, \ldots, Q_m \in A$ such that $Q_1 >_\Gamma Q_2 >_\Gamma \ldots >_\Gamma Q_m$.*

Proof. Since r_i connects N_{i-1} with N_i for $i = 1, \ldots, m$, we know by Definition 4.2 that $\overleftarrow{r_i} \subseteq \mathsf{cons}(N_{i-1})$ and $\mathsf{cons}(N_i) \subseteq \overrightarrow{r_i}$. This implies that $\overleftarrow{r_i} \subseteq \overrightarrow{r_{i-1}}$ for all $i, 1 < i \leq m$.

For each of the edges r_i we have assumed that it is new, i.e., $r_i \notin A_0$. Thus, r_i must have been added by some rule application. Condition 3 of Definition 4.3 entails then that, for every $1 \leq i \leq m$, $\overrightarrow{r_i} \cap \mathsf{cons}(A_0) = \emptyset$, and thus, for every $1 < i \leq m$ it also holds that $\overleftarrow{r_i} \cap \mathsf{cons}(A_0) = \emptyset$, as $\overleftarrow{r_i} \subseteq \overrightarrow{r_{i-1}}$.

Since r_m was added by a rule application, by Condition 2 of Definition 4.3, there must be an assertion $Q_m \in A \cap \widehat{\Lambda}$ such that $\overleftarrow{r_m} \subseteq \mathsf{cons}(Q_m)$. Hence, it is the case that $\mathsf{cons}(Q_m) \not\subseteq \mathsf{cons}(A_0)$. By Lemma 4.7, there exist $r \in A \cap \widehat{\Delta}$ and $Q_{m-1} \in A \cap \widehat{\Lambda}$ such that $\overleftarrow{r} \subseteq \mathsf{cons}(Q_{m-1}), \mathsf{cons}(Q_m) \subseteq \overrightarrow{r}$, and $Q_m <_\Gamma Q_{m-1}$. We have $\overleftarrow{r_m} \subseteq \overrightarrow{r_{m-1}}$ and $\overleftarrow{r_m} \subseteq \mathsf{cons}(Q_m) \subseteq \overrightarrow{r}$, which implies that $\overrightarrow{r_{m-1}} \cap \overrightarrow{r} \neq \emptyset$. However, Conditions 3 and 4 of Definition 4.3 ensure that distinct assertions in $\widehat{\Delta} \setminus A_0$ must have disjoint sets of descendants. Thus, we know that $r = r_{m-1}$.

We can now apply the same argument to r_{m-1} and Q_{m-1} and obtain an assertion Q_{m-2} such that $\overleftarrow{r_{m-2}} \subseteq \mathsf{cons}(Q_{m-2}), \mathsf{cons}(Q_{m-1}) \subseteq \overrightarrow{r_{m-2}}$, and $Q_{m-1} <_\Gamma Q_{m-2}$. By iterating this argument, we thus obtain the desired descending chain of assertions $Q_1 >_\Gamma Q_2 >_\Gamma \ldots >_\Gamma Q_m$. □

The following two remarks will be useful in the proof of the main theorem of this section. First, recall that Condition 7 of Definition 4.3 ensures that the assertions from $\widehat{\Lambda}$ triggering a rule application all belong to the same node.

Second, given a new node N (i.e., one that was not present in the initial state) and an assertion $P \in N$, Lemma 4.7 yields an edge r such that $\mathsf{cons}(P) \subseteq \overrightarrow{r}$. Since distinct edges have disjoint sets of descendants (Condition 4 of Definition 4.3) any other assertion in $Q \in N$ also satisfies $\mathsf{cons}(Q) \subseteq \overrightarrow{r}$. This shows that the constants occurring in a node all belong to the descendant set of the edge whose introduction created the node.

We are now ready to show termination of the pinpointing extension of any ordered forest tableaux.

Theorem 4.9. *If S is an ordered forest tableau, then its pinpointing extension terminates on every input.*

Proof. Suppose that there is an input $\Gamma = (\mathcal{I}, \mathcal{T})$ for which there is an infinite sequence of pinpointing rule applications $\mathfrak{S}_0 \rightarrow_{S^{pin}} \mathfrak{S}_1 \rightarrow_{S^{pin}} \ldots$, with $\mathfrak{S}_0 \in \Gamma^S$. Since S is a

covered tableau, there is a finite $\mathcal{T}$-cover Ω_Γ such that, for all $i \geq 0$, the assertions in $\mathfrak{S}_i$ use only predicate symbols from Ω_Γ. As noted above, for every node there is a fixed finite set of constants that can occur in the assertions of this node. This set is either the set of constants occurring in $\mathfrak{S}_0$ (for an old node) or it consists of the descendants in the unique edge whose introduction created the node (for a new node). Together with the fact that the $\mathcal{T}$-cover Ω_Γ is finite, this restricts the assertions that can occur in the node to a fixed finite set. Each of these assertion may repeatedly have its label modified by applications of the pinpointing rules. However, every application of a rule makes the label more general in the sense that the new monotone Boolean formula has more models than the previous one. Since these formulae are built over a finite set of propositional variables, this can happen only finitely often. The same argument shows that the label of a given edge can be changed only finitely often.

Hence, to get a non-terminating sequence of rule applications, infinitely many new nodes must be added. By Conditions 5 and 2 of Definition 4.3, each newly added node N is created as successor of an existing node w.r.t. a unique edge $r \in \widehat{\Delta}$ such that the constants in N are new constants contained in $\overrightarrow{r}$. If infinitely many new nodes are created, then either there is a node that obtains infinitely many direct successors, or an infinite chain of nodes is created, where each is a successor of the previous one.

Proposition 4.8 implies that the latter case cannot occur. In fact, given nodes $N_0, N_1, \ldots, N_m$ and edges $r_1, \ldots, r_m$ such that, for all $i, 1 \leq i \leq m$, r_i connects N_{i-1} to N_i, Proposition 4.8 yields a sequence of assertions $Q_1, \ldots, Q_m \in \widehat{\Lambda}$ such that $Q_1 >_\Gamma Q_2 >_\Gamma \ldots >_\Gamma Q_m$. However, the length of such a descending sequence is bounded by the cardinality of the finite $\mathcal{T}$-cover Ω_Γ. Thus, it is not possible that an infinite path is created by a sequence of rule applications.

Now, consider the first case, i.e., assume that there is a node N for which infinitely many successors are created. However, the constants in N are from a fixed finite set of constants C, and the predicate symbols that can occur in the applied rules must all belong to the finite $\mathcal{T}$-cover Ω_Γ. Thus, up to variable renaming, there are only finitely many rules that can be applied to N, and there are only finitely many ways of replacing the variables in the left-hand side of rules by constants from C. The fresh variables in the right-hand side are always replaced by distinct new constants. Thus, for a fixed rule and a fixed substitution σ replacing the variables in the left-hand side of this rules by constants from C, the assertions introduced by two different applications of this rule using σ only differ by a renaming of these new constants. By the way pinpointing rule applicability is defined, such renamed variants can only be added as long as their labels are not equivalent. But there are only finitely many labels up to equivalence. Thus, N can in fact obtain only a finite number of successors. This finishes the proof that the pinpointing extension of an ordered forest tableau always terminates. □

Note that termination of the pinpointing extension implies termination of the original tableau. In fact, a non-terminating sequence of rule applications for the original tableau can easily be transformed into a non-terminating sequence of rule applications for its pinpointing extension.

Corollary 4.10. *An ordered forest tableau terminates on every input.*

The definition of forest tableaux imposes quite a number of restrictions to be satisfied. Thus, it is natural to ask whether all these restrictions are indeed necessary. The answer is yes: if any of these restrictions is removed, then Theorem 4.9 no longer holds. In fact, it is possible to construct tableaux satisfying all other properties that do not terminate. More interesting perhaps is that there are *terminating* tableaux satisfying all other properties whose pinpointing extensions do not terminate. Here, we illustrate this fact with one example, where we remove Condition 6 of Definition 4.3. Examples for the other conditions can be built in a similar way.

Example 4.11. *Consider the tableau S that has the following four rules:*

$$\begin{aligned}
\mathsf{R}_1 &: (\{P(x)\}, \{ax_1\}) \to \{\{R(x), Q_1(x)\}\},\\
\mathsf{R}_2 &: (\{P(x)\}, \{ax_2\}) \to \{\{R(x), Q_2(x)\}\},\\
\mathsf{R}_3 &: (\{R(x)\}, \emptyset) \to \{\{r(x,y)\}, \{Q_1(x)\}, \{Q_2(x)\}\},\\
\mathsf{R}_4 &: (\{P(x), r(x,y)\}, \emptyset) \to \{\{T(y), r(x,z)\}\},
\end{aligned}$$

and where the function $\cdot^S$ maps every input $\mathcal{I} \in \mathfrak{I}$ to the singleton set $\{\{P(a)\}\}$, and each axiom in $\mathfrak{T} = \{ax_1, ax_2\}$ to the empty set.

It is easy to verify that S with the ordering $T < Q_2 < Q_1 < R < P$ satisfies all the conditions of an ordered forest tableau, except for Condition 6 of Definition 4.3 violated by the rule R_4.

For any axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$, we have $\Gamma^S = \{(\{P(a)\}, \mathcal{T})\}$, and thus neither R_3 nor R_4 is applicable to Γ^S. Depending on which axioms are contained in $\mathcal{T}$, the rules R_1 and/or R_2 may be applicable. However, their application introduces $Q_1(a)$ or $Q_2(a)$ into the set of assertions, and thus the non-deterministic rule R_3 is not applicable. Obviously, R_4 becomes applicable only after R_3 has been applied. Consequently, S terminates on every axiomatised input Γ.

It is possible, however, to construct an infinite chain of pinpointing rule applications starting with $\Gamma^S = \{(\{P(a)\}, \{ax_1, ax_2\})\}$ where $\mathsf{lab}(P(a)) = \top$. In fact, we can first apply the rule R_1. This adds the assertions $R(a)$ and $Q_1(a)$, both with label ax_1. An application of the rule R_2 adds the assertion $Q_2(a)$ with label ax_2, and modifies the label of the assertion $R(a)$ to $\mathsf{lab}(R(a)) = ax_1 \vee ax_2$. At this point, we have reached an S-state $\mathfrak{S}$ containing the assertions $P(a)$, $R(a)$, $Q_1(a)$, $Q_2(a)$ with labels $\mathsf{lab}(P(a)) = \top$, $\mathsf{lab}(R(a)) = ax_1 \vee ax_2$, $\mathsf{lab}(Q_1(a)) = ax_1$, and $\mathsf{lab}(Q_2(a)) = ax_2$. The rule R_3 is pinpointing applicable to this S-state. Indeed, although both $Q_1(a)$ and $Q_2(a)$ are contained in the assertion set of $\mathfrak{S}$, their labels are not implied by $\mathsf{lab}(R(a))$. The application of R_3 to $\mathfrak{S}$ replaces $\mathfrak{S}$ by three new S-states. One of these new S-states contains the assertion $r(a,b)$ for a new constant b. At this point, rule R_4 becomes applicable. Its application adds the assertions $T(b)$ and $r(a,c)$ for a new constant c. Since there is no assertion of the form $T(c)$, R_4 becomes again applicable, and its application adds a new constant d within an assertion $r(a,d)$. It is easy to see that we can now continue applying rule R_4 indefinitely. ■

Finding a non-terminating tableau is an easier task. If we consider the tableau that has only the rule R_4 and where every input $\mathcal{I} \in \mathfrak{I}$ is mapped to $\{\{P(a), r(a,b)\}\}$, then

this yields an example of a non-terminating tableau that satisfies all the conditions of an ordered forest tableau, except for Condition 6.

4.3 Blocking in Forest Tableaux

The ordered forest tableaux introduced in the previous section can be used to model tableau-based algorithms that try to generate a finite tree- or forest-shaped model. In the presence of so-called general concept inclusion axioms (GCIs) or transitive roles, DLs lose the finite tree/forest model property, and thus these algorithms need no longer terminate. Termination can be regained, however, by blocking the application of generating rules, i.e., rules that generate new nodes, in case that the node to which the rule is supposed to be applied has a predecessor node that has the same assertions. A saturated and clash-free tableau can then be unraveled into an infinite tree/forest model (see, e.g., [HS99]).

In order to illustrate our general model of tableaux with blocking, we consider a non-terminating forest tableau that can be made terminating by blocking. Note that the usual tableau-based algorithm for unsatisfiability of $\mathcal{ALC}$ concepts w.r.t. $\mathcal{SI}$-TBoxes shows a similar behavior (see Section 2.3.5).

Example 4.12. *Consider a forest tableau S with the following three (deterministic) rules*

$$\begin{aligned} \mathsf{R}_1 &: (\{C(x)\}, \emptyset) \to \{\{r(x,y), D(y)\}\}, \\ \mathsf{R}_2 &: (\{D(x)\}, \emptyset) \to \{\{r(x,y), C(y)\}\}, \\ \mathsf{R}_3 &: (\{C(x), r(y,x)\}, \emptyset) \to \{\{\neg D(y)\}\}, \end{aligned}$$

and the clash $\{D(x), \neg D(x)\}$. In addition, we assume that the function $\cdot^S$ maps every input $\mathcal{I} \in \mathfrak{I}$ to the singleton set $\{\{C(a_0)\}\}$ and each axiom in $\mathfrak{T}$ to the empty set. It is easy to see that S does not terminate since it can produce an infinite chain of assertions of the form $C(a_0), r(a_0,a_1), D(a_1), r(a_1,a_2), C(a_2), \ldots$. If we apply rule R_1 followed by R_2 to $\Gamma^S = \{(\{C(a_0)\}, \emptyset)\}$, then we obtain the S-state $(A, \emptyset)$ consisting of the assertions $A := \{C(a_0), r(a_0,a_1), D(a_1), r(a_1,a_2), C(a_2)\}$. At this point, blocking should prevent the application of R_1 to the node a_2:[14] it is the repeated application of R_1 that causes the generation of the above infinite chain of assertions. The reason why R_1 can be blocked is that the node a_2 contains the same assertions as its predecessor a_0: both have an assertion for C (see Figure 4.1). Note, however, that the application of R_3 to a_2, which adds the assertion $\neg D(a_1)$, should still be possible. In fact, otherwise the clash could not be detected. After rule R_3 has been applied to this S-state, we reach the S-state $(A \cup \{\neg D(a_1)\}, \emptyset)$ depicted in Figure 4.2, where the only applicable rule is R_1, which is however blocked. Thus, the blocking variant of the tableau terminates with this blocking-saturated state. ■

The difference between the rules R_1 and R_3 that makes the latter applicable while the former is blocked is that an application of R_1 adds new constants. Only this kind of

[14] Since in this forest tableau the elements of Λ are all unary, nodes are uniquely identified by constants.

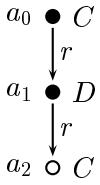

Figure 4.1: Rule R_1 is blocked

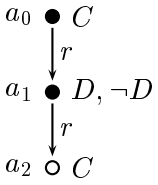

Figure 4.2: Blocking-saturated S-state

rules will be blocked, while non-generating rules will always be applicable, regardless of the relationships between the nodes at the S-state.

Before we can formalise our notion of tableaux with blocking, we need to introduce some notation. In the following we always assume that we have a forest tableau S. Given an input Γ, any S-state that can be generated from Γ^S by the applications of the rules of S is called an *S-state for* Γ. We now assume that all the S-states that we consider are S-state for some input.

The rule $(B_0, \mathcal{S}) \rightarrow \{B_1, \ldots, B_m\}$ is called *generating* if there is an $i, 1 \leq i \leq m$, such that $B_i \cap \widehat{\Delta} \neq \emptyset$. Note that the definition of forest tableaux implies that, if such a generating rule is applicable with substitution ρ in state $\mathfrak{S}$, then $\mathfrak{S}$ contains a (unique) node N such that $B_0\rho \subseteq N$. We can thus talk about *the* node to which a generating rule is applicable and/or applied. Given an S-state $\mathfrak{S}$ for the input Γ, a node N in $\mathfrak{S}$ is *new* if it has been generated by the application of a generating rule. Note that this is the case iff $\mathsf{cons}(N) \cap \mathsf{cons}(\Gamma^S) = \emptyset$. Only new nodes will be allowed to be blocked.

Given two nodes N, N', we say that they contain the same assertions (written $N \equiv N'$) if there is a bijection $f : \mathsf{cons}(N) \rightarrow \mathsf{cons}(N')$ such that $P(a_1, \ldots, a_n) \in N$ iff $P(f(a_1), \ldots, f(a_n)) \in N'$.

Definition 4.13 (Blocking). *Given a forest tableau S, and an axiomatised input Γ, let $\mathfrak{S}$ be an S-state for Γ. The* blocking relation $\lhd$ *between nodes of $\mathfrak{S}$ is defined as follows:*

$$N_1 \lhd N_2 \quad \textit{iff} \quad N_1 \equiv N_2, N_2 \textit{ is a predecessor of } N_1, \textit{and } N_1 \textit{ is a new node.}$$

The node N is blocked *if either there is a node N' such that $N \lhd N'$, or the parent node of N is blocked. A non-generating rule is* $\lhd$-applicable *if it is applicable in the sense of Definition 3.16; a generating rule is* $\lhd$-applicable *if it is applicable and the node N to which it is applicable is not blocked.*

For sets of S-states $\mathcal{M}, \mathcal{M}'$ (S-states $\mathfrak{S}, \mathfrak{S}'$) we write $\mathcal{M} \rightarrow_S^{\lhd} \mathcal{M}'$ ($\mathfrak{S} \rightarrow_S^{\lhd} \mathfrak{S}'$) if $\mathcal{M} \rightarrow_S \mathcal{M}'$ ($\mathfrak{S} \rightarrow_S \mathfrak{S}'$) using a rule that is $\lhd$-applicable. The set of S-states $\mathcal{M}$ is $\lhd$-saturated *if there is no $\mathcal{M}'$ such that $\mathcal{M} \rightarrow_S^{\lhd} \mathcal{M}'$.* ■

In Figures 4.1 and 4.2 the node a_2 is blocked by the node a_0, which we represent with an unfilled circle. The notion of correctness of blocking tableaux is analogous to the one for general tableaux from the previous chapter.

Definition 4.14 (Correctness). *Let $\mathcal{P}$ be a c-property on axiomatised inputs for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$, and S a forest tableau for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$. Then S is* $\lhd$-correct for $\mathcal{P}$ *if it terminates and is sound and complete with respect to $\lhd$-application, i.e., the following two conditions hold for every axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$:*

1. *there is no infinite chain of rule applications $\Gamma^S = \mathcal{M}_0 \to_S^{\lhd} \mathcal{M}_1 \to_S^{\lhd} \ldots$;*
2. *for every chain of rule applications $\Gamma^S = \mathcal{M}_0 \to_S^{\lhd} \ldots \to_S^{\lhd} \mathcal{M}_n$ such that $\mathcal{M}_n$ is $\lhd$-saturated we have that $\Gamma \in \mathcal{P}$ iff $\mathcal{M}_n$ is full of clashes.*

■

In the DL literature, different forms of blocking have been used. The variant that we model here is usually called *equality blocking* [HS99] since it requires that the blocked and the blocking nodes have the same set of assertions. In *subset blocking* [BBH96], it is only required that the blocking node has all the assertions of the blocked node, but not necessarily *vice versa*. Our reason for using equality blocking rather than subset blocking is that it is more appropriate for DLs with inverse roles, and our notion of forest tableaux can model tableau-based algorithms for DLs with inverse roles. DLs that have both inverse roles and number restrictions require more complex notions of blocking, such as *pair-wise blocking* [HST00], that look not just at one node but at a node and its neighbors. Since our current notion of tableaux does not capture rules that can identify distinct constants to represent the same individual, as used in tableau-based algorithms for DLs with number restrictions [HB91], we have decided not to model pair-wise blocking.

The notion of blocking introduced in Definition 4.13 ensures that every covered forest tableau terminates with respect to $\lhd$-application on all inputs. Instead of showing this directly, we will prove that this is the case even for its pinpointing extension. But first, we must adapt the notion of blocking to the pinpointing extension. Obviously, this notion must take the labels of assertions into account as well.

Given an input Γ, any S-state that can be generated from Γ^S by the applications of the rules of the pinpointing extension of S is called a *labeled S-state for* Γ. Nodes of such a labeled S-state will be called *labeled nodes*. Given two such labeled nodes N, N', we say that they contain the same labeled assertions (written $N \equiv_{pin} N'$) if there is a bijection $f : \mathsf{cons}(N) \to \mathsf{cons}(N')$ such that $P(a_1, \ldots, a_n) \in N$ iff $P(f(a_1), \ldots, f(a_n)) \in N'$, and the labels of these assertions, $\mathsf{lab}(P(a_1, \ldots, a_n))$ and $\mathsf{lab}(P(f(a_1), \ldots, f(a_n)))$ are (propositionally) equivalent.

Definition 4.15 (Pinpointing blocking). *Given a forest tableau S, and an axiomatised input Γ, let $\mathfrak{S}$ be a labeled S-state for Γ. The* blocking relation $\lhd_{pin}$ *between labeled nodes of $\mathfrak{S}$ is defined as follows:*

$N_1 \lhd_{pin} N_2$ *iff* $N_1 \equiv_{pin} N_2$, *N_2 is a predecessor of N_1, and N_1 is a new node.*

The node N is pinpointing blocked *if either there is a node N' such that $N \lhd_{pin} N'$, or the parent node of N is pinpointing blocked.*

We define the notions $\lhd_{pin}$-applicable and $\lhd_{pin}$-application as well as $\to_{S^{pin}}^{\lhd}$ and $\lhd_{pin}$-saturated in the obvious way. ■

Our approach for proving termination of the pinpointing extension of a covered forest tableau with respect to $\lhd_{pin}$-application is similar to the one employed for showing that ordered forest tableaux always terminate. Equipped with Lemma 4.4, we can prove the desired termination result.

Theorem 4.16. *Let S be a covered forest tableau. Then the pinpointing extension of S terminates with respect to $\lhd_{pin}$-application on every input.*

Proof. Suppose that there is an input $\Gamma = (\mathcal{I}, \mathcal{T})$ for which there is an infinite sequence of pinpointing rule applications $\mathfrak{S}_0 \to_{S^{pin}} \mathfrak{S}_1 \to_{S^{pin}} \cdots$, where $\mathfrak{S}_0 \in \Gamma^S$. Since S is a covered tableau, there is a finite $\mathcal{T}$-cover Ω_Γ such that the assertions in $\mathfrak{S}_i$ use only predicate symbols from Ω_Γ, for every $i \geq 0$. As already noted, every node has a fixed finite set of constants that can appear in its assertions. By Lemma 4.4, this set is either the set of constants occurring in $\mathfrak{S}_0$ (for an old node) or the descendants in the unique edge by which the node was created (for a new node). Since the $\mathcal{T}$-cover is finite, the assertions that can occur in a given node form a finite set. Each of these assertions may repeatedly have its label modified by pinpointing rule applications; however, every pinpointing rule application produces a more general label, in the sense that the new monotone Boolean formula has more models than the previous one. Since these formulas are built over a finite set of propositional variables, this can happen only finitely often. Analogously, the label of a given edge can be changed only finitely often.

Hence, to produce a non-terminating sequence of rule applications, infinitely many new nodes must be added. Conditions 5 and 2 of Definition 4.3 ensure that every newly added node N is created as a successor of an existing node with a unique edge $r \in \widehat{\Delta}$ connecting them, and all the constants in N are new constants appearing in $\overrightarrow{r}$. If infinitely many new nodes are created, then either there is a node with infinitely many direct successors, or an infinite chain of nodes, each one being a successor of the previous, is created. The first case can be treated as in the proof of Theorem 4.9.

Thus, we concentrate on the second case. The number of constants occurring in a new node is bounded by the largest arity of a predicate name $r \in \widehat{\Delta}$. Taking into account that there are also only finitely many possible labels, this implies that there can only be finitely many different labeled nodes, up to constant renaming. Then, for every chain of nodes $N_0, N_1, \ldots, N_m$ that is sufficiently long (i.e., where m is larger than the maximal number of labeled nodes that are different up to constant renaming), there must exist $1 \leq k < \ell \leq m$ such that $N_k \equiv_{pin} N_\ell$, and thus N_ℓ is pinpointing blocked by N_k. Consequently, all the nodes N_r for $r > \ell$ are pinpointing blocked, which in particular means that N_m cannot get a successor node. Thus, the second case is not possible either, which completes the proof of the theorem. □

As in the case of ordered tableaux, termination of the pinpointing extension also implies termination of the original tableau, as stated by the following corollary.

Corollary 4.17. *Let S be a covered forest tableau. Then S terminates with respect to $\lhd$-application on every input.*

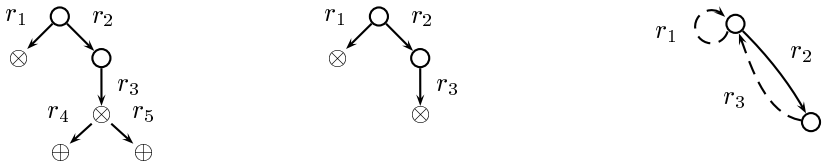

Figure 4.3: Example of folding of an S-state

It is worth noticing here that the tableau from Example 4.11 is also an instance of terminating tableaux whose pinpointing extension does not terminate, even when using blocking. This is the case since, for this particular example, the violation of Condition 6 of Definition 4.3 leads to a node that has infinitely many direct successors, hence producing an infinite tree, even though its depth is finitely bounded.

We have seen that blocking can be used to regain termination of non-terminating covered forest tableaux, and that this is also the case for the pinpointing extension. However, since blocking prevents the application of rules that would be applicable in the normal sense, the proof of correctness of the pinpointing extension given in Section 3.3 does not apply directly to the pinpointing extension of tableaux with blocking. A new proof is hence necessary.

Our proof of correctness will rely on the notion of the *folded version* of an S-state, which is obtained by removing all blocked nodes and adding new edges. Let S be a forest tableau and $\mathfrak{S} = (A, \mathcal{T})$ an S-state for an input Γ. Then $\mathfrak{S}$ is a *forest-structure*, i.e., it is a graph-structure consisting of a set of tree-like structures growing out of the original graph-structure induced by the input. If we remove all the blocked nodes that are descendants of other blocked nodes, we obtain a new forest-structure $\mathfrak{S}' = (A', \mathcal{T})$ in which blocked nodes appear only as leafs in the trees. For every pair of nodes N_1 and N_2 in $\mathfrak{S}'$, if N_1 is blocked by N_2, then we know that $N_1 \equiv N_2$, and hence there is a bijection $f : \mathsf{cons}(N_1) \to \mathsf{cons}(N_2)$ such that $P(a_1, \ldots, a_n) \in N_1$ iff $P(f(a_1), \ldots, f(a_n)) \in N_2$. We modify the edge with destination N_1 (i.e., the unique assertion $r(\overleftarrow{r}, \overrightarrow{r}) \in \widehat{\Delta} \cap A'$ with $\mathsf{cons}(N_1) \subseteq \overrightarrow{r}$) to $r(\overleftarrow{r}, f(\overrightarrow{r}))$ and then remove N_1.[15] Since $f(\overrightarrow{r})$ contains only constants from N_2, this new edge points to N_2, i.e., to the node that blocks N_1. By applying this modification for all the remaining blocked nodes, we obtain the *folded version* of $\mathfrak{S}$, which we denote by $\mathfrak{S}^{\circlearrowleft}$. If $\mathcal{M}$ is a set of S-states, then its *folded version* is $\mathcal{M}^{\circlearrowleft} = \{\mathfrak{S}^{\circlearrowleft} \mid \mathfrak{S} \in \mathcal{M}\}$. Figure 4.3 shows the process of folding an S-state. The tree in the left shows the tree shape of an S-state, where the two nodes marked as $\otimes$ are blocked by the root node, and the nodes marked as $\oplus$ are blocked since their parent node is blocked. When we remove the latter ones, we obtain a tree where only leafs have blocked nodes (center). Finally, these blocked nodes are removed, and the previous edges leading to them are modified to lead to the root node that was blocking them, represented as dashed arcs on the right-most graph.

Let us illustrate folding of S-states in a more concrete way, using the tableau of Example 4.12. We have seen there that rule application can be used to obtain the

[15] We denote as $f(\overrightarrow{r})$ the tuple obtained by applying the function f to each element of $\overrightarrow{r}$.

$\lhd$-saturated S-state $\mathfrak{S} = (A, \mathcal{T})$ where

$$A = \{C(a_0), r(a_0, a_1), D(a_1), \neg D(a_1), r(a_1, a_2), C(a_2)\}.$$

The folded version of this S-state does not contain the constant a_2 (since the blocked node $\{C(a_2)\}$ has been removed), but it makes up for this by an edge from a_1 to a_0; in other words, $\mathfrak{S}^{\circlearrowleft} = (A^{\circlearrowleft}, \mathcal{T})$ with $A^{\circlearrowleft} = \{C(a_0), r(a_0, a_1), D(a_1), \neg D(a_1), r(a_1, a_0)\}$.

The next lemma will allow us to reuse some of the results shown in Section 3.3, by relating $\lhd$-saturatedness of a state to "normal" saturatedness of the corresponding folded state.

Lemma 4.18. *If $\mathfrak{S}$ is $\lhd$-saturated, then $\mathfrak{S}^{\circlearrowleft}$ is saturated.*

Proof. Let $\mathfrak{S} = (A, \mathcal{T})$, $\mathfrak{S}^{\circlearrowleft} = (A^{\circlearrowleft}, \mathcal{T})$ and $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$ be applicable to $\mathfrak{S}^{\circlearrowleft}$ with substitution ρ. Assume first that R is a generating rule, and let N be the node in $A^{\circlearrowleft}$ to which this rule is applied, i.e., $B_0\rho \subseteq N \subseteq A^{\circlearrowleft}$. Since folding never modifies any nodes in the graph structure, except from removing some, N is also a node in $\mathfrak{S}$, i.e., $B_0\rho \subseteq N \subseteq A$. As $\mathfrak{S}$ is $\lhd$-saturated, R is not $\lhd$-applicable to it. This means that either N is blocked, or there is a substitution σ extending ρ such that $B_i\sigma \subseteq A$ for some $i, 1 \leq i \leq m$. Since folding removes all blocked nodes and N belongs to $A^{\circlearrowleft}$, the first case cannot occur; thus, the second option must be the case. We can then construct a substitution σ' extending ρ such that $B_i\sigma' \subseteq A^{\circlearrowleft}$ as follows: for every $x \in \bigcup_{j=0}^{m} \mathsf{var}(B_j)$, if $\sigma(x)$ is a constant in a non-blocked node of A, then we define $\sigma'(x) := \sigma(x)$; if $\sigma(x)$ belongs to a node N_1 blocked by some non-blocked node N_2, then in particular $N_1 \equiv N_2$, and thus there exists a bijection $f : \mathsf{cons}(N_1) \to \mathsf{cons}(N_2)$ such that $P(a_1, \ldots, a_n) \in N_1$ iff $P(f(a_1), \ldots, f(a_n)) \in N_2$; in this case, we define $\sigma'(x) = f(\sigma(x))$. Because these bijections are also used when defining the folded state, it is easy to see that $B_i\sigma' \subseteq A^{\circlearrowleft}$ indeed holds. This contradicts our assumption that R is applicable to $\mathfrak{S}^{\circlearrowleft}$ with substitution ρ.

Suppose now that R is a non-generating rule. If $B_0\rho \subseteq A$, since $\lhd$-applicability coincides with regular applicability for non-generating rules, the proof is analogous to the one for the previous case. Thus, we can assume w.l.o.g. that $B_0\rho \not\subseteq A$. Then, $B_0\rho$ must contain edges r that were added by the folding process; these edges are of the form $r = p(\overleftarrow{r}, f_r(\overrightarrow{r}))$ where f_r is the bijection ensuring equivalence between the blocked and the blocking nodes, and there are corresponding edges in A that have blocked nodes as destinations. Using the bijections f_r to rename constants, we can define a substitution ρ' such that $B_0\rho' \subseteq A$. Note that this inclusion depends on our use of equality blocking. In fact, an assertion $P\rho \in B_0\rho$ may be an assertion in a blocking node N, whose constants are renamed in ρ' such that they belong to a node N' blocked by N. Thus, we need to know that all the assertions occurring in the blocking node also occur (appropriately renamed) in the blocked node. This is guaranteed by our definition of $\equiv$.

Since $\mathfrak{S}$ is $\lhd$-saturated, R is not applicable to $\mathfrak{S}$ with substitution ρ', which implies that there must exist an $i, 1 \leq i \leq m$ such that $B_i\rho' \subseteq A$. We claim that $B_i\rho \subseteq A^{\circlearrowleft}$. This is an easy consequence of the facts that (i) the assertions of non-blocked nodes in A are contained also in $A^{\circlearrowleft}$; and (ii) the assertions of blocked nodes in A are contained

in a renamed variant in the blocking node (i.e., the node to which the edge leading to the blocked node has been redirected). □

As we did for the case without blocking in Section 3.3, we will use projections of labeled S-states to show the correctness of the pinpointing extension. The next lemma states a close connection between pinpointing $\lhd$-saturatedness of a set of labeled S-states and $\lhd$-saturatedness of its projection.

Lemma 4.19. *Let $\mathcal{M}$ be a finite set of labeled S-states and $\mathcal{V}$ a propositional valuation. If $\mathcal{M}$ is pinpointing $\lhd$-saturated, then $\mathcal{V}(\mathcal{M})$ is $\lhd$-saturated.*

Proof. Suppose that there is an S-state $\mathfrak{S} = (A, \mathcal{T}) \in \mathcal{M}$ and a rule of the form $\mathsf{R} : (B_0, \mathcal{S}) \to \{B_1, \ldots, B_m\}$ such that R is $\lhd$-applicable to $\mathcal{V}(\mathfrak{S})$ with substitution ρ. For non-generating rules, applicability and $\lhd$-applicability coincide. Consequently, if R in non-generating, then we can re-use the proof of Lemma 3.13, which shows the result for the case without blocking.

Thus, assume that R is a generating rule. We have that $\mathcal{S} \subseteq \mathcal{T}_\mathcal{V}, B_0\rho \subseteq A_\mathcal{V}$, for every $i, 1 \leq i \leq m$ and every substitution ρ' on $\mathsf{var}(B_0 \cup B_i)$ extending ρ, it holds that $B_i\rho' \not\subseteq A_\mathcal{V}$, and the node N' in $\mathcal{V}(\mathfrak{S})$ to which the rule is applied is not blocked.

We will show now that R is pinpointing $\lhd$-applicable to $\mathfrak{S}$ with the same substitution ρ. Since $\mathcal{S} \subseteq \mathcal{T}_\mathcal{V} \subseteq \mathcal{T}$ and $B_0\rho \subseteq A_\mathcal{V} \subseteq A$, the first two conditions of pinpointing applicability are satisfied. For the third condition, consider an i and a substitution ρ' on $\mathsf{var}(B_0 \cup B_i)$ extending ρ. We must show that $\mathsf{ins}(B_i\rho', A) \neq \emptyset$ where $\psi = \bigwedge_{b \in B_0} \mathsf{lab}(b\rho) \wedge \bigwedge_{s \in \mathcal{S}} \mathsf{lab}(s)$. Note that $\mathcal{S} \subseteq \mathcal{T}_\mathcal{V}$ and $B_0\rho \subseteq A_\mathcal{V}$ imply that $\mathcal{V}$ satisfies ψ. Since $B_i\rho' \not\subseteq A_\mathcal{V}$, there is a $b \in B_i$ such that $b\rho' \notin A_\mathcal{V}$. Thus $b\rho' \notin A$ or $\mathcal{V}$ does not satisfy $\mathsf{lab}(b\rho')$. In the first case, $b\rho'$ is clearly ψ-insertable into A. In the second case, $\psi \not\models \mathsf{lab}(b\rho')$ since $\mathcal{V}$ satisfies ψ, and thus $b\rho'$ is again ψ-insertable into A.

We have shown up to now that R is pinpointing applicable to $\mathfrak{S}$ with the substitution ρ. It remains to show that the node $N \subseteq A$ to which this rule is applicable (i.e., the node satisfying $B_0\rho \subseteq N \subseteq A$) is not pinpointing blocked. If N is not a new node, then it cannot be blocked. Thus, we can restrict the attention to the case where N is a new node. Since $B_0\rho \subseteq A_\mathcal{V}$, we have $B_0\rho \subseteq N_\mathcal{V}$. Thus, the node N' in $\mathcal{V}(\mathfrak{S})$ to which the rule R is applied is a subset of $N_\mathcal{V}$.[16] We know that this node is not blocked. Also note that, since this node belongs to $\mathcal{V}(\mathfrak{S})$, the sequence of edges in $\mathfrak{S}$ that leads to the node N is also contained in $\mathcal{V}(\mathfrak{S})$ and leads to this node. In fact, the label of an edge is always implied by the labels of assertions occurring in nodes or as edges below this edge.

Assume that N is pinpointing blocked. We concentrate on the case where there is a predecessor node M of N such that $M \equiv_{pin} N$. (The case where the parent node of N is blocked can be reduced to this case by considering, instead of N, the (unique) predecessor node N' of N that is blocked, but whose parent node is not blocked.) The definition of the relation $\equiv_{pin}$ implies that there is a bijection f such that, for every assertion $P(a_1, \ldots, a_n) \in N' \subseteq N_\mathcal{V}$ the assertion $P(f(a_1), \ldots, f(a_n)) \in M_\mathcal{V}$. The fact that the assertions in N' are connected implies that their f-images in $M_\mathcal{V}$ are also

[16] Note that connectedness of N need not imply connectedness of $N_\mathcal{V} \subseteq N$.

connected, and thus they belong to a node $M' \subseteq M_{\mathcal{V}}$. This shows, however, that N' is blocked by M', which is a contradiction. □

Notice that if $\mathcal{M} \to_S^{\lhd} \mathcal{M}'$, then it is also the case that $\mathcal{M} \to_S \mathcal{M}'$, and analogously for pinpointing rule application: if $\mathcal{M} \to_{S^{pin}}^{\lhd} \mathcal{M}'$, then $\mathcal{M} \to_{S^{pin}} \mathcal{M}'$. This, along with (2) of Lemma 3.26, shows that $\mathcal{M} \to_{S^{pin}}^{\lhd} \mathcal{M}'$ implies that either $\mathcal{V}(\mathcal{M}) \to_{S^m} \mathcal{V}(\mathcal{M}')$ or $\mathcal{V}(\mathcal{M}) = \mathcal{V}(\mathcal{M}')$. In particular, $\mathcal{M}_0 \overset{*}{\to}_{S^{pin}}^{\lhd} \mathcal{M}$ implies $\mathcal{V}(\mathcal{M}_0) \overset{*}{\to}_{S^m} \mathcal{V}(\mathcal{M})$.

One last observation before proceeding to the proof of correctness of the pinpointing extension is that the order in which rules are applied has no influence on the result of a blocking tableau.

Lemma 4.20. *Let Γ be an axiomatised input, and $\mathcal{M}_0 = \Gamma^S$. If there are $\mathcal{M}$ and $\mathcal{M}'$ such that $\mathcal{M}_0 \overset{*}{\to}_S \mathcal{M}$ and $\mathcal{M}_0 \overset{*}{\to}_S \mathcal{M}'$ and $\mathcal{M}, \mathcal{M}'$ are both $\lhd$-saturated, then $\mathcal{M}$ is full of clashes iff $\mathcal{M}'$ is also full of clashes.*

Proof. For every S-state $\mathfrak{S} \in \mathcal{M}'$, there is an S-state $\mathfrak{S}_0 \in \mathcal{M}_0$ such that $\mathfrak{S}_0 \preceq \mathfrak{S}$, where the corresponding constant renaming function is the identity. Recall that folding only changes assertions involving blocked nodes, and that only new nodes can be blocked. Consequently, we also have $\mathfrak{S}_0 \preceq \mathfrak{S}^{\circlearrowleft}$. Since $\mathfrak{S}^{\circlearrowleft}$ is saturated by Lemma 4.18, Lemma 3.30 thus yields an S-state $\mathfrak{S}' \in \mathcal{M}$ such that $\mathfrak{S}' \preceq \mathfrak{S}^{\circlearrowleft}$.

Now, assume that $\mathcal{M}$ is full of clashes, i.e., every element of $\mathcal{M}$ contains a clash. To show that $\mathcal{M}'$ is full of clashes, consider $\mathfrak{S} \in \mathcal{M}'$. Then there is an element $\mathfrak{S}' \in \mathcal{M}$ such that $\mathfrak{S}' \preceq \mathfrak{S}^{\circlearrowleft}$. Since $\mathcal{M}$ is full of clashes, $\mathfrak{S}'$ contains a clash, and thus $\mathfrak{S}^{\circlearrowleft}$ also contains a clash. Since $\mathfrak{S}^{\circlearrowleft}$ is obtained from $\mathfrak{S}$ by removing blocked nodes and changing some edges, and since clashes consider only single nodes, this implies that $\mathfrak{S}$ also contains a clash.

The other direction can be shown analogously. □

Theorem 4.21 (Correctness of pinpointing with blocking). *Let S be a forest tableau for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$ that is $\lhd$-correct for the c-property $\mathcal{P}$. Then the following holds for every axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$ over $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$:*

> *For every chain of rule applications $\mathcal{M}_0 \to_{S^{pin}}^{\lhd} \ldots \to_{S^{pin}}^{\lhd} \mathcal{M}_n$ such that $\mathcal{M}_0 = \Gamma^S$ and $\mathcal{M}_n$ is pinpointing $\lhd$-saturated, the clash formula $\psi_{\mathcal{M}_n}$ induced by $\mathcal{M}_n$ is a pinpointing formula for $\mathcal{P}$ and Γ.*

Proof. Let $\Gamma = (\mathcal{I}, \mathcal{T})$ be an axiomatised input, and assume that $\Gamma^S = \mathcal{M}_0 \overset{*}{\to}_{S^{pin}}^{\lhd} \mathcal{M}_n$ with $\mathcal{M}_n$ pinpointing $\lhd$-saturated. To show that $\psi_{\mathcal{M}_n}$ is a pinpointing formula for $\mathcal{P}$, we have to show that, for every propositional valuation $\mathcal{V}$, it holds that $(\mathcal{I}, \mathcal{T}_{\mathcal{V}}) \in \mathcal{P}$ iff $\mathcal{V}$ satisfies $\psi_{\mathcal{M}_n}$.

Let $\mathcal{N}_0 = (\mathcal{I}, \mathcal{T}_{\mathcal{V}})^S$. Since S terminates w.r.t. $\lhd$-application, there is a $\lhd$-saturated set $\mathcal{N}$ such that $\mathcal{N}_0 \overset{*}{\to}_S^{\lhd} \mathcal{N}$. Also, since $\mathcal{M}_0 \overset{*}{\to}_{S^{pin}}^{\lhd} \mathcal{M}_n$, it must be the case that $\mathcal{V}(\mathcal{M}_0) \overset{*}{\to}_{S^m} \mathcal{V}(\mathcal{M}_n)$. Additionally, $\mathcal{V}(\mathcal{M}_0) = \mathcal{N}_0$ and also $\mathcal{V}(\mathcal{M}_n)$ is $\lhd$-saturated. Thus, Lemma 4.20 yields that $\mathcal{N}$ is full of clashes iff $\mathcal{V}(\mathcal{M}_n)$ is full of clashes. By the $\lhd$-correctness of S for $\mathcal{P}$, we have then that $(\mathcal{I}, \mathcal{T}_{\mathcal{V}}) \in \mathcal{P}$ iff $\mathcal{N}$ is full of clashes iff $\mathcal{V}(\mathcal{M}_n)$ is full of clashes iff $\mathcal{V}$ satisfies $\psi_{\mathcal{M}_n}$ (Lemma 3.12). □

Our notion of $\lhd$-correctness explicitly requires termination w.r.t. $\lhd$-application. For covered forest tableaux we have seen that this condition is always satisfied.

Corollary 4.22. *Let S be a covered forest tableau for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$ that is sound and complete w.r.t. $\lhd$-application, i.e., for every chain of rule applications of the form $\mathcal{M}_0 \to_S^{\lhd} \ldots \to_S^{\lhd} \mathcal{M}_n$ such that $\mathcal{M}_0 = \Gamma^S$ and $\mathcal{M}_n$ is $\lhd$-saturated we have that $\Gamma \in \mathcal{P}$ iff $\mathcal{M}_n$ is full of clashes. Then the following holds for every axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$ over $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$:*

1. *There is no infinite chain of rule applications $\Gamma^S = \mathcal{M}_0 \to_{S^{pin}}^{\lhd} \mathcal{M}_1 \to_{S^{pin}}^{\lhd} \ldots$;*
2. *For every chain of rule applications $\Gamma^S = \mathcal{M}_0 \to_{S^{pin}}^{\lhd} \ldots \to_{S^{pin}}^{\lhd} \mathcal{M}_n$ such that $\mathcal{M}_n$ is pinpointing $\lhd$-saturated, the clash formula $\psi_{\mathcal{M}_n}$ induced by $\mathcal{M}_n$ is a pinpointing formula for $\mathcal{P}$ and Γ.*

In this chapter we presented some restrictions that force a tableau to produce states that have a forest-like structure. If we additionally bound the set of predicate names that can be used in the construction of states to be finite, we obtain forest structures with finite branching. In order to ensure termination, we require also that the structures have a finite depth. We showed two ways to achieve this. The first one is by obtaining a partial ordering on the predicate names such that every rule application produces only *smaller* assertions. The second method consists on changing the applicability conditions of rules in order to implement a blocking mechanism. The blocking mechanism used in this work follows the ideas of what is called *equality blocking* in the DL literature, as it is triggered only if the blocking- and blocked-nodes have both equivalent assertions. The approach followed clearly shows that blocking imposes additional difficulties for defining the pinpointing extension, and for proving its correctness.

In the following chapter we will leave behind the tableau-based approach towards deciding a property and focus on another prominent method; namely, the automata-based approach. We will show that it is possible to find a pinpointing formula for a property that is decided by a so-called *axiomatic automaton*. Furthermore, since decisions in this method are based on an emptiness test that can be performed in finite time, we do not have to deal with the termination problems presented by the tableaux approach. Perhaps more interesting is that the extension for finding a pinpointing formula is also terminating, and actually requires only polynomial time on the size of the original automaton.

Chapter 5

Automata-based Pinpointing

In this chapter we leave behind the tableau-based approach and focus on automata-based decision procedures. In a nutshell, we will show that if we can decide a property $\mathcal{P}$ with an automata-based method, then we can also compute a pinpointing formula for $\mathcal{P}$. As an additional advantage, we will show that the computation of this pinpointing formula can be done in time polynomial in the size of the automaton that decides $\mathcal{P}$.

The automata-based approach differs from the tableau-based in the way the decisions are made. Intuitively, we can think of the rule application in general tableaux as an attempt to build a model that verifies (or falsifies) the property being tested; on the other hand, the iterative emptiness test used by the automata-based approach can be seen as an attempt to prove the (non-)existence of such a model, without actually building it. In other words, tableau-based decision procedures can be seen as *constructive* proofs of the fact that the given axiomatised input belongs to the property, as opposed to the *non-constructive* proofs obtained by means of automata. Consider for instance unsatisfiability of $\mathcal{ALC}$ concept terms w.r.t. $\mathcal{SI}$-TBoxes. An axiomatic input $(C, \mathcal{T})$ belongs to this c-property if and only if there is no model $\mathcal{I}$ of the TBox $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. The tableau-based decision procedure tries to falsify this condition by forcing an interpretation to map the concept term C to a non-empty set, and then expanding it to satisfy all the conditions required from a model. Only if this construction terminates without finding a contradiction is the input rejected (see Section 2.3.5). The automata-based decision procedure for the same c-property reduces the problem to deciding the existence of a run of a looping automaton whose root is labeled with an initial state. But the emptiness test does not try to construct such a run; instead, it finds the set of all states that can serve as root for runs of the automaton, and compares it with the set of initial states (see Section 2.4.1). At no point of this process is the actual construction of a run attempted.

While a non-constructive approach is certainly enough for deciding a property, where we want only to test whether a model exists, it is not completely obvious how these ideas can generalise to the computation of a pinpointing formula, or in general MinAs and MaNAs for the axiomatised input and decided property. Basically, with a constructive approach we can also highlight the specific steps that need to be executed for adding a specific piece to the model, as we did in the pinpointing

extension of general tableaux (Chapter 3). Having a non-constructive proof disallows the application of this method. It is in that respect that this chapter introduces a novel idea, showing that not only constructive decision procedures can be extended to labeling methods that ultimately compute a pinpointing formula. Our approach makes the assumptions that individual axioms have an influence in the construction of the automaton that is independent of the presence or absence of other axioms, and that we can represent this influence by restricting the transition relation and initial states from a weaker automaton. Although these assumptions clearly affect the generality of the method, we believe that they are reasonable, and still allow for deciding and pinpointing several c-properties of interest.

The chapter is divided as follows. We first show how any automaton deciding a c-property can be transformed into a weighted automaton whose so-called behaviour corresponds to the pinpointing formula. We then present an iterative procedure for computing the behaviour of weighted automata over any finite distributive lattice; the automaton used for pinpointing being a special case covered by this algorithm. During the development of our work, an alternative algorithm for computing the behaviour of weighted tree automata working on infinite trees was independently developed in [DKR08]. We devote the last section of this chapter to a comparison of the two algorithms, with a special emphasis on their application to pinpointing.

5.1 Pinpointing Automata

As mentioned already in repeated opportunities, automata can also be used to decide properties in DLs and other logics. In the case of the algorithm presented in Section 2.4.1, the c-property under consideration is *un*satisfiability of a concept term w.r.t. a general $\mathcal{SI}$-TBox. Likewise, in Section 2.4.2, we decide the c-property of axiomatic *un*satisfiability of LTL formulae. The decision procedures consisted on performing an emptiness test on the automaton $\mathcal{A}^{\mathsf{sat}}_{C,\mathcal{T}}$ (see Definition 2.19) or $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}}$ (Definition 2.23). The property under consideration holds if and only if the automaton has no successful run whose root is labeled with an initial state.

Contrary to the tableau-based approach presented in Chapter 3, the axioms are not used explicitly for deciding the property, but are only implicit in the construction of the automaton. For instance, the TBox is used to define the transition relation of the automaton $\mathcal{A}^{\mathsf{sat}}_{C,\mathcal{T}}$ by restricting the set of usable transitions to only those that were compatible with it. In the automaton $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}}$, the LTL formulae in the set $\mathcal{R}$ restrict the set of initial states. If the axiomatised input belongs to the property being decided by such an automaton, it is impossible to distinguish the axioms that are relevant for this fact from those that are superfluous, and thus, the only possible way to compute the set of MinAs and MaNAs is by trial and error, constructing one automaton for each possible subset of axioms and performing the emptiness test on it.

In general, the automata-based approach for deciding a property $\mathcal{P}$ consists on translating each axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$ into an automaton $\mathcal{A}_\Gamma$ such that $\Gamma \in \mathcal{P}$ if and only if $\mathcal{A}_\Gamma$ has *no* successful runs. Since we want to find out how the axioms relate to each other with respect to the c-property under consideration, we need to

know how the absence of some of the axioms in $\mathcal{T}$ would influence the construction of the automaton. We thus assume that for every $\mathcal{T}' \subseteq \mathcal{T}$, the automaton $\mathcal{A}_{(\mathcal{I},\mathcal{T}')}$ can be constructed from $\mathcal{A}_\Gamma$ by appropriately restricting its set of transitions and initial states. To this end we will employ two so-called *restricting functions*.

Definition 5.1 (Restricting functions, restricted automaton). *Let $\mathcal{A}$ be the generalised Büchi automaton $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$ for arity k and $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatised input. The functions $\Delta\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q^{k+1})$ and $I\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q)$ are called a* transition restricting function *and an* initial restricting function*, respectively. We extend these restricting functions to be applicable over sets of axioms as follows:*

$$\begin{aligned} \Delta\mathsf{res}(\mathcal{T}') &:= \bigcap_{t \in \mathcal{T}'} \Delta\mathsf{res}(t) \text{ and} \\ I\mathsf{res}(\mathcal{T}') &:= \bigcap_{t \in \mathcal{T}'} I\mathsf{res}(t). \end{aligned}$$

*If $\mathcal{T}' \subseteq \mathcal{T}$, then the $\mathcal{T}'$-*restricted subautomaton of $\mathcal{A}$ w.r.t. $\Delta\mathsf{res}$ and $I\mathsf{res}$ *is the generalised Büchi automaton $\mathcal{A}_{|\mathcal{T}'}$ defined as*

$$\mathcal{A}_{|\mathcal{T}'} := (Q, \Delta \cap \Delta\mathsf{res}(\mathcal{T}'), I \cap I\mathsf{res}(\mathcal{T}'), F_1, \ldots, F_n).$$

■

We will give the name of *axiomatic automata* to generalised Büchi tree automata equipped with a transition- and an initial-restricting function.

Definition 5.2 (Axiomatic automaton). *Let $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$ be a generalised Büchi automaton for arity k, $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatised input, and the functions $\Delta\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q^{k+1})$ and $I\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q)$ a transition and an initial restricting function, respectively. The tuple $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ is called an* axiomatic automaton *for* Γ. ■

An axiomatic automaton is considered correct for a property $\mathcal{P}$ if the restricted subautomata decide $\mathcal{P}$ for the axiomatised input corresponding to each subset of axioms.

Definition 5.3 (Correctness). *Given a c-property $\mathcal{P}$, $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ is* correct *for Γ w.r.t. $\mathcal{P}$ if for every $\mathcal{T}' \subseteq \mathcal{T}$ it is the case that $(\mathcal{I}, \mathcal{T}') \in \mathcal{P}$ iff the restricted subautomaton $\mathcal{A}_{|\mathcal{T}'}$ has no successful run r such that $r(\varepsilon) \in I \cap I\mathsf{res}(\mathcal{T}')$.* ■

Consider again the automaton $\mathcal{A}^{\mathsf{sat}}_{C,\mathcal{T}}$ defined in Section 2.4.1. This automaton correctly decides unsatisfiability w.r.t. general $\mathcal{SI}$-TBoxes but still lack appropriate restricting functions, a necessary condition in the definition of axiomatic automata. It is easy to notice that the only place where the axioms influence the construction of this automaton is in the transition relation Δ, which is defined as the set of all tuples in Q^{k+1} that satisfy the Hintikka condition and are compatible with all the

axioms in $\mathcal{T}$. Thus, we can alternatively remove the second condition in the definition of this transition relation, that is, the condition of compatibility with the TBox, and obtain the same intended behaviour through the transition restricting function. Since in this case the axioms do not influence the set of initial states, we can set the function $I\mathsf{res}_{C,\mathcal{T}}$ as the constant function Q; i.e., the function that maps every axiom in $\mathcal{T}$ to the set of all states Q.

Definition 5.4 (Axiomatic automaton for $\mathcal{SI}$). *Let C be a concept term, $\mathcal{T}$ a general $\mathcal{SI}$-TBox and k the number of existential restrictions in $\mathsf{sub}(C,\mathcal{T})$. The axiomatic automaton $(\mathcal{A}_{C,\mathcal{T}}, \Delta\mathsf{res}_{C,\mathcal{T}}, I\mathsf{res}_{C,\mathcal{T}})$ has as its first component the looping automaton $\mathcal{A}_{C,\mathcal{T}} = (Q, \Delta, I)$ where*

- *Q is the set of all Hintikka sets for $(C,\mathcal{T})$;*
- *Δ is the set of all tuples $(H_0, H_1, \ldots, H_k) \in Q^{k+1}$ satisfying the Hintikka condition; and*
- *$I = \{H \in Q \mid C \in H\}$.*

The transition restricting function $\Delta\mathsf{res}_{C,\mathcal{T}}$ maps each axiom $t \in \mathcal{T}$ to the set of all tuples in Δ that are compatible with t. The initial restricting function $I\mathsf{res}_{C,\mathcal{T}}$ maps each axiom $t \in \mathcal{T}$ to the set Q. ■

One can see that for $\mathcal{T}' \subseteq \mathcal{T}$, the $\mathcal{T}'$-restricted subautomaton of $\mathcal{A}_{C,\mathcal{T}}$ is exactly the automaton $\mathcal{A}^{\mathsf{sat}}_{C,\mathcal{T}'}$. Thus, this construction yields a correct axiomatic automaton for unsatisfiability of $\mathcal{ALC}$ concept terms w.r.t. $\mathcal{SI}$-TBoxes.

Theorem 5.5. *Let C be an $\mathcal{ALC}$ concept term and $\mathcal{T}$ a general $\mathcal{SI}$-TBox. The axiomatic automaton $(\mathcal{A}_{C,\mathcal{T}}, \Delta\mathsf{res}_{C,\mathcal{T}}, I\mathsf{res}_{C,\mathcal{T}})$ is correct for $(C,\mathcal{T})$ w.r.t. unsatisfiability.*

To obtain an axiomatic automaton for axiomatic unsatisfiability of LTL formulae, we can follow a similar idea. Notice that, in this case, the axioms have no impact on the transition relation of the automaton $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}}$, but rather in the set of initial states. Thus, we can weaken the definition of $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}}$ such that its set of initial states is now given by all elementary sets that contain the static formula ϕ. Since we do not want axioms to affect the transition relation of the restricted automaton, we set, for every $\psi \in \mathcal{R}$, the transition restricting function $\Delta\mathsf{res}_{\phi,\mathcal{R}}(\psi) = \Delta$. The initial restricting function $I\mathsf{res}_{\phi,\mathcal{R}}$ then maps every LTL formula $\psi \in \mathcal{R}$ to the set of elementary sets containing ψ.

Definition 5.6 (Axiomatic automaton for LTL). *Let ϕ and $\mathcal{R}$ be an LTL formula and a set of LTL formulae, respectively, and let $\theta_1 \mathcal{U} \psi_1, \ldots, \theta_n \mathcal{U} \psi_n$ be all the until formulae in $\mathsf{cl}(\phi,\mathcal{R})$. The axiomatic automaton $(\mathcal{A}_{\phi,\mathcal{R}}, \Delta\mathsf{res}_{\phi,\mathcal{R}}, I\mathsf{res}_{\phi,\mathcal{R}})$ has as its first component the generalised Büchi automaton $\mathcal{A}_{\phi,\mathcal{R}} := (Q, \Delta, I, F_1, \ldots, F_n)$, where*

- *Q is the set of all elementary sets for $(\phi,\mathcal{R})$;*
- *Δ consists of all compatible pairs $(H, H') \in Q \times Q$;*

- $I := \{H \in Q \mid \phi \in H\}$;
- $F_i := \{H \in Q \mid \psi_i \in H \text{ or } \theta_i \mathcal{U} \psi_i \notin H\}$.

For every $\psi \in \mathcal{R}$, the transition restricting and initial restricting functions are given by $\Delta\mathsf{res}_{\phi,\mathcal{R}}(\psi) := \Delta$ and $I\mathsf{res}_{\phi,\mathcal{R}}(\psi) := \{H \in Q \mid \psi \in H\}$, respectively. ■

Clearly, for every $\mathcal{R}' \subseteq \mathcal{R}$, the $\mathcal{R}'$-restricted subautomaton of $\mathcal{A}_{\phi,\mathcal{R}}$ is equivalent to the automaton $\mathcal{A}^{\mathsf{sat}}_{\phi,\mathcal{R}'}$. This means that the axiomatic automaton constructed this way is correct for $(\phi, \mathcal{R})$ w.r.t. axiomatic unsatisfiability.

Theorem 5.7. *Let ϕ and $\mathcal{R}$ be an LTL formula and a set of LTL formulae, respectively. The axiomatic automaton $(\mathcal{A}_{\phi,\mathcal{R}}, \Delta\mathsf{res}_{\phi,\mathcal{R}}, I\mathsf{res}_{\phi,\mathcal{R}})$ is correct for $(\phi, \mathcal{R})$ w.r.t. axiomatic unsatisfiability.*

Given an axiomatic automaton that correctly decides a c-property, we will construct a weighted automaton whose so-called behaviour corresponds to the pinpointing formula for this property. Weighted automata do not merely accept or reject an input tree, but rather assign a value to it; these values come from a distributive lattice [Grä98].

Definition 5.8 (Distributive lattice). *A* distributive lattice *is a partially ordered set $(S, \leq_S)$ such that infima and suprema of arbitrary finite subsets of S always exist and distribute over each other. The distributive lattice $(S, \leq_S)$ is called* finite *if its carrier set S is finite.* ■

As we will see next, any weighted automaton uses as weights only finitely many elements of the underlying distributive lattice. Since finitely generated distributive lattices are finite [Grä98], the closure of this set under the lattice operations infimum and supremum yields a finite distributive lattice. For this reason, we will in the following assume without loss of generality that the weights of our weighted Büchi automaton come from a *finite* distributive lattice $(S, \leq_S)$.

For the rest of this chapter, we will often simply use the carrier set S to denote the distributive lattice $(S, \leq_S)$. The infimum (supremum) of a subset $T \subseteq S$ will be denoted by $\bigotimes_{t \in T} t$ ($\bigoplus_{t \in T} t$). We will often compute the infimum (supremum) $\bigotimes_{i \in I} t_i$ ($\bigoplus_{i \in I} t_i$) over an infinite set of indices I. However, the finiteness of the lattice and the idempotency of the operators infimum and supremum ensure that the sets over which the operators are actually applied are finite, and hence infimum and supremum are well-defined in this case. For the infimum (supremum) of two elements, we will also use the infix notation; i.e., write $t_1 \otimes t_2$ ($t_1 \oplus t_2$) to denote the infimum (supremum) of the set $\{t_1, t_2\}$. The least element of S (i.e., the infimum of the whole set S) will be denoted by $\mathbf{0}$, and the greatest element (i.e., the supremum of the whole set S) by the symbol $\mathbf{1}$.

It should be noted that our assumption that the weights come from a finite distributive lattice is stronger than the one usually encountered in the literature on weighted automata. In fact, for automata working on finite trees, it is sufficient to assume that the weights come from a so-called semiring [Sei94]. In order to have a

well-defined behaviour also for weighted automata working on infinite objects, the existence of infinite products and sums is required [DR06, Rah07]. The additional properties imposed by our requirement to have a finite distributive lattice (in particular, the idempotency of product and sum) will be used to show that we can actually compute the behaviour of our weighted Büchi automata (see Section 5.2).[17] Since our main goal in the use of weighted automata is to compute a pinpointing formula, these stronger assumption will not be problematic. As we will see later, the weights used for computing this formula actually belong to a finitely generated free distributive lattice.

Definition 5.9 (Weighted Büchi automaton). *Let S be a finite distributive lattice. A* weighted generalised Büchi automaton (WGBA) *over S for arity k is a tuple of the form $\mathcal{A} = (Q, \mathsf{in}, \mathsf{wt}, F_1, \ldots, F_n)$ where:*

- *Q is a finite set of* states,
- *$\mathsf{in} : Q \to S$ is the* initial distribution,
- *$\mathsf{wt} : Q^{k+1} \to S$ assigns* weights *to transitions, and*
- *$F_1, \ldots, F_n \subseteq Q$ are the sets of* final states.

A WGBA is called weighted Büchi automaton (WBA) *if $n = 1$ and* weighted looping automaton (WLA) *if $n = 0$.*

A run *of a WGBA $\mathcal{A}$ is a labeled tree $r : K^* \to Q$. The* weight *of this run is $\mathsf{wt}(r) = \bigotimes_{u \in K^*} \mathsf{wt}(\overrightarrow{r(u)})$. This run is* successful *if for every path p and every $i, 1 \leq i \leq n$, there are infinitely many nodes $u \in \mathsf{p}$ such that $r(u) \in F_i$. Let $\mathsf{succ}_{\mathcal{A}}$ denote the set of all successful runs of $\mathcal{A}$. The* behaviour *of the automaton $\mathcal{A}$ is*

$$\|\mathcal{A}\| := \bigoplus_{r \in \mathsf{succ}_{\mathcal{A}}} \mathsf{in}(r(\varepsilon)) \otimes \mathsf{wt}(r).$$

■

For example, the Boolean semiring $\mathbb{B} = (\{0, 1\}, \wedge, \vee, 1, 0)$ is a finite distributive lattice, where the partial order is defined as $1 \leq_{\mathbb{B}} 0$. Note that we have defined 1 to be smaller than 0, and thus in this context conjunction yields the supremum (i.e., is the "addition" $\oplus$) and disjunction yields the infimum (i.e., is the "product" $\otimes$). Likewise, 1 is the least element $\mathbf{0}$, and 0 is the greatest element $\mathbf{1}$. Any generalised Büchi tree automaton $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$ can easily be transformed into a WGBA $\mathcal{A}_w$ on $\mathbb{B}$ such that the behaviour of $\mathcal{A}_w$ is 0 iff $\mathcal{A}$ has a successful run. In $\mathcal{A}_w$, the initial distribution maps initial states to 0 and all other states to 1; a tuple in Q^{k+1} receives weight 0 if it belongs to Δ, and weight 1 otherwise. We will now see that this automaton behaves just as it was previously claimed.

[17] Alternatively to the idempotency assumption, one can try to ensure convergence of these infinitary operators with the help of a so-called discounting function [DK06, Man08, DSV08]. Since we want axioms to have the same influence over the result, regardless on where in the model they are used, we will not follow these ideas.

The emptiness test for Büchi automata sketched in Section 2.4 can be adapted such that it computes the behaviour of $\mathcal{A}_w$ as follows. We will construct a function $\mathsf{bad} : Q \to \{0,1\}$ such that $\mathsf{bad}(q) = 1$ iff q is a bad state. The outer iteration of the algorithm will update this function at every step. In the beginning, no state is known to be bad, and thus we start the iteration with $\mathsf{bad}_0(q) = 0$ for all $q \in Q$. Now assume that the function $\mathsf{bad}_i : Q \to \{0,1\}$, for $i \geq 0$, has already been computed. For the next step of the iteration, we call the inner loop to update the set of adequate states. In this loop, we are going to compute the function $\mathsf{adq}^i : Q \to \{0,1\}$. Here, $\mathsf{adq}^i(q) = 1$ means that q is *not* an adequate state, i.e., that it is not possible to construct a run with q at the root where each path reaches at least one final state. At the beginning we know nothing about the adequate states, so we set $\mathsf{adq}^i_0(q) = 1$ for all $q \in Q$. Assume that we have already computed $\mathsf{adq}^i_n : Q \to \{0,1\}$. To know whether a state should become adequate in the next step, we need to check for each transition starting from this state whether the final states reached by the transition are non-bad, and the non-final states are already known to be adequate. Thus, we have

$$\mathsf{adq}^i_{n+1}(q) = \bigwedge_{(q,q_1,\ldots,q_k)\in Q^{k+1}} \mathsf{wt}(q,q_1,\ldots,q_k) \vee \bigvee_{q_j \notin F} \mathsf{adq}^i_n(q_j) \vee \bigvee_{q_j \in F} \mathsf{bad}_i(q_j). \quad (5.1)$$

The function adq^i is the limit of this inner iteration, which is reached after at most $|Q|$ steps. With this function, we define

$$\mathsf{bad}_{i+1}(q) = \mathsf{bad}_i(q) \vee \mathsf{adq}^i(q).$$

The function bad is the limit of this outer iteration, which is also reached after at most $|Q|$ steps. This computation of the function bad by two nested iterations basically simulates the computation of all bad states in the emptiness test for Büchi tree automata that we sketched in Section 2.4. It is thus easy to show that $\mathsf{bad}(q) = 1$ iff q is a bad state, i.e., cannot occur as a label in a successful run of $\mathcal{A}$.

Given the definition of $\mathcal{A}_w$, it is easy to see that a run $r : K^* \to Q$ of $\mathcal{A}_w$ has weight 0 iff it is a run of $\mathcal{A}$ that starts with an initial state of $\mathcal{A}$. Consequently, $\mathcal{A}$ has a successful run that starts with an initial state iff

$$\|\mathcal{A}_w\| = \bigwedge_{r \in \mathsf{succ}_{\mathcal{A}_w}} \mathsf{in}(r(\varepsilon)) \vee \mathsf{wt}(r) = 0.$$

Putting these observations together, we thus have: the behaviour of $\mathcal{A}_w$ is 0 iff $\mathcal{A}$ has a successful run that starts with an initial state iff there is an initial state q (i.e., $\mathsf{in}(q) = 0$) that is not bad (i.e., $\mathsf{bad}(q) = 0$). This shows that the behaviour of $\mathcal{A}_w$ is given by $\bigwedge_{q\in Q} \mathsf{in}(q) \vee \mathsf{bad}(q)$. Later, we will see that the behaviour of a WBA can always be computed by such a procedure with two nested iterations.

Starting from a correct axiomatic automaton, we can construct a weighted automaton whose behaviour corresponds exactly to a pinpointing formula. Obviously, the semiring used by this automaton needs to have monotonic Boolean formulae as elements. We use the $\mathcal{T}$-Boolean semiring. Recall that every axiom in $\mathcal{T}$ is labeled

with a unique propositional variable, and $\mathsf{lab}(\mathcal{T})$ represents the set of all the labels of elements in $\mathcal{T}$. The *$\mathcal{T}$-Boolean semiring* is given by $\mathbb{B}^{\mathcal{T}} = (\hat{\mathbb{B}}(\mathcal{T}), \wedge, \vee, \top, \bot)$, where $\hat{\mathbb{B}}(\mathcal{T})$ is the quotient set of all monotonic Boolean formulae over $\mathsf{lab}(\mathcal{T})$ by the propositional equivalence relation; in other words, two propositionally equivalent formulae correspond to the same element in $\hat{\mathbb{B}}(\mathcal{T})$. This semiring is indeed a distributive lattice, where the partial order is defined as $\phi \leq \psi$ iff $\psi \to \phi$ is a valid propositional formula. Furthermore, as $\mathcal{T}$ is a finite set of axioms, this lattice is also finite: it corresponds to the free distributive lattice over the generators $\mathsf{lab}(\mathcal{T})$. Note that, similar to the case of the Boolean semiring $\mathbb{B}$ defined above, conjunction is the semiring addition (i.e., yields the supremum $\oplus$) and disjunction is the semiring multiplication (i.e., yields the infimum $\otimes$). Likewise, $\top$ is the least element $\mathbf{0}$ and $\bot$ is the greatest element $\mathbf{1}$.

Definition 5.10 (Pinpointing automaton). *Let $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ be an axiomatic automaton for the axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$, with $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$. The* violating functions $\Delta\mathsf{vio} : Q^{k+1} \to \mathbb{B}^{\mathcal{T}}$ *and* $I\mathsf{vio} : Q \to \mathbb{B}^{\mathcal{T}}$ *are given by*

$$\Delta\mathsf{vio}(q_0, q_1, \ldots, q_k) := \bigvee_{\{t \in \mathcal{T} \mid (q_0, q_1, \ldots, q_k) \notin \Delta\mathsf{res}(t)\}} \mathsf{lab}(t); \text{ and}$$

$$I\mathsf{vio}(q) := \bigvee_{\{t \in \mathcal{T} \mid q \notin I\mathsf{res}(t)\}} \mathsf{lab}(t).$$

The pinpointing automaton *induced by $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ w.r.t. $\mathcal{T}$ is the WGBA $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}$ over $\mathbb{B}^{\mathcal{T}}$, given by $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin} = (Q, \mathsf{in}, \mathsf{wt}, F_1, \ldots, F_n)$, where*

$$\mathsf{in}(q) = \begin{cases} I\mathsf{vio}(q) & \text{if } q \in I \\ \top & \text{otherwise;} \end{cases}$$

$$\mathsf{wt}(q, q_1, \ldots, q_k) = \begin{cases} \Delta\mathsf{vio}(q, q_1, \ldots, q_k) & \text{if } (q, q_1, \ldots, q_k) \in \Delta \\ \top & \text{otherwise.} \end{cases}$$

■

Let r be a tree labeled with elements of Q. It is easy to see that if r corresponds to a run of the automaton $\mathcal{A}$, then its weight when seen as a run of $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}$ is $\mathsf{wt}(r) = \bigvee_{u \in \mathsf{K}^*} \Delta\mathsf{vio}(\overrightarrow{r(u)})$; on the contrary case, its weight is $\mathsf{wt}(r) = \top$. Intuitively, the violating function $\Delta\mathsf{vio}$ expresses which axioms are *not satisfied* – or "violated" – by a given transition. The weight of a run accumulates then all the axioms violated by any of the transitions appearing as labels in this run. Additionally, the function $I\mathsf{vio}$ represents the axioms that are violated by the initial state of the run. Thus, removing all the axioms appearing in these two formulae would yield a subset of axioms that are not violated by this run. This means that, if the run is successful and the root is labeled with an initial state, due to correctness, the property does not hold anymore after the removal of those axioms. But different runs may lead to different sets of axioms that need to be removed, and hence we need the conjunction of all of them to obtain a pinpointing formula.

Theorem 5.11. *Let $\mathcal{P}$ be a c-property, and $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatised input. If the axiomatic automaton $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ is correct for Γ w.r.t. $\mathcal{P}$, then the behaviour $\|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}\|$ is a pinpointing formula for Γ w.r.t. $\mathcal{P}$.*

Proof. We need to show that, for every valuation $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$, it holds that $\mathcal{V}$ satisfies $\|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}\|$ iff $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$. Let $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$ be an arbitrary valuation.

Suppose first that $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \notin \mathcal{P}$. Since $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ is correct for Γ w.r.t. $\mathcal{P}$, there must be a successful run r of $\mathcal{A}_{|\mathcal{T}_\mathcal{V}}$ with $r(\varepsilon) \in I \cap I\mathsf{res}(\mathcal{T}_\mathcal{V})$. Consequently, $\overrightarrow{r(u)} \in \Delta\mathsf{res}(\mathcal{T}_\mathcal{V})$ holds for every $u \in K^*$, and thus $\mathcal{V}$ cannot satisfy $\Delta\mathsf{vio}(\overrightarrow{r(u)})$, for any $u \in K^*$. Since r is a successful run of $\mathcal{A}_{|\mathcal{T}_\mathcal{V}}$, it is also a successful run of $\mathcal{A}$, which implies $\mathsf{wt}(r) = \bigvee_{u \in K^*} \Delta\mathsf{vio}(\overrightarrow{r(u)})$. Thus, $\mathcal{V}$ does not satisfy $\mathsf{wt}(r)$. Since $r(\varepsilon) \in I$, we know that $\mathsf{in}(r(\varepsilon)) = I\mathsf{vio}(r(\varepsilon))$; additionally, $r(\varepsilon) \in I\mathsf{res}(\mathcal{T}_\mathcal{V})$ implies that $\mathcal{V}$ does not satisfy $I\mathsf{vio}(r(\varepsilon))$. Thus, $\mathcal{V}$ does not satisfy $\mathsf{in}(r(\varepsilon)) \vee \mathsf{wt}(r)$. But then $\mathcal{V}$ also cannot satisfy the conjunctive formula $\bigwedge_{r \in \mathsf{succ}} \mathsf{in}(r(\varepsilon)) \vee \mathsf{wt}(r) = \|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}\|$.

Conversely, if $\mathcal{V}$ does not satisfy $\|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}\| = \bigwedge_{r \in \mathsf{succ}} \mathsf{in}(r(\varepsilon)) \vee \mathsf{wt}(r)$, then there must exist a successful run r such that $\mathcal{V}$ does not satisfy $\mathsf{in}(r(\varepsilon)) \vee \mathsf{wt}(r)$. This implies that $r(\varepsilon) \in I \cap I\mathsf{res}(\mathcal{T}_\mathcal{V})$ and that $\overrightarrow{r(u)} \in \Delta\mathsf{res}(\mathcal{T}_\mathcal{V})$ for all $u \in K^*$. Consequently, r is a successful run of $\mathcal{A}_{|\mathcal{T}_\mathcal{V}}$ with $r(\varepsilon) \in I \cap I\mathsf{res}(\mathcal{T}_\mathcal{V})$, which shows $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \notin \mathcal{P}$, by the correctness of the axiomatic automaton. □

This theorem shows that it suffices to compute the behaviour of the pinpointing automaton $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}$ induced by an axiomatic automaton $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ in order to obtain a pinpointing formula for the property decided by $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$. When we began this work, we were unable to find any algorithm for computing the behaviour of weighted automata in the literature and hence had to develop our own, which generalises the ideas used in the iterative emptiness test of unweighted automata (Section 2.4). During the development of our work, an alternative algorithm for computing the behaviour of weighted tree automata working on infinite trees has independently been developed in [DKR08]. It turns out, however, that using this algorithm in our pinpointing application basically yields a so-called black-box approach for pinpointing, in which the set of all MinAs is obtained by testing for emptiness of the restricted subautomaton defined by each subset of axioms. The pinpointing formula in disjunctive normal form is then obtained from this set as described by the Expression 3.2 in page 37. Instead, our algorithm tries to compute the pinpointing formula within a time bound proportional to the one required for a single emptiness test. We describe this in more detail in the following sections.

5.2 Computing the Behaviour of Weighted Automata

In this section, we first show how the behaviour of a weighted Büchi automaton over a finite distributive lattice can be computed by two nested iterations. We then show how, if we restrict the discourse to WLAs, the procedure can be simplified to a single bottom-up iteration. Afterwards, we prove that for every WGBA one can construct in polynomial time a WBA having the same behaviour, thus obtaining a method for computing the behaviour of WGBAs also in polynomial time. This latter

reduction follows the ideas that have previously been used for the case of unweighted automata [VW86].

5.2.1 Computing the Behaviour of a WBA

By definition, the behaviour of a weighted Büchi automaton is the addition of the weights of all successful runs, which themselves consist of the product of the weights of all transitions that they contain, multiplied by the initial distribution of their root labels. Trying to apply this definition directly to the computation of the behaviour will unavoidably lead to failure given the potentially infinite number of successful runs and the infinite size of each of them. To overcome this problem, we will generalise the iterative algorithm for deciding emptiness of Büchi automata that was sketched in Section 2.4 and produce a method that computes the behaviour in a similar fashion. To introduce the ideas, we will consider a Büchi automaton as a WBA over the Boolean semiring as described in page 80. The two iterations described there, namely the one that computes the functions adq^i (Equation 5.1) and the one that computes the function bad, will be generalised to monotone operators that can be applied to arbitrary finite distributive lattices.

For the remaining of this section we will assume that we have an arbitrary but fixed WBA $\mathcal{A} = (Q, \mathsf{in}, \mathsf{wt}, F)$ over the finite distributive lattice S. We will show that $\mathcal{A}$ induces a monotone operator $\mathcal{Q} : S^Q \to S^Q$, where S^Q is the set of all mappings from Q to S, and that the behaviour of $\mathcal{A}$ can easily be obtained from the greatest fixpoint of this operator. The partial order $\leq_S$ can be transferred to S^Q in the usual way, by applying it component-wise: if $\sigma, \sigma' \in S^Q$, then $(\sigma \oplus \sigma')(q) = \sigma(q) \oplus \sigma'(q)$ for all $q \in Q$. It is easy to see that $(S^Q, \leq_{S^Q})$ is again a finite distributive lattice. We will use $\otimes$ and $\oplus$ also to denote the infimum and supremum in S^Q. The least (respectively greatest) element of S^Q is the function $\widetilde{\mathbf{0}}$ (respectively $\widetilde{\mathbf{1}}$) that maps every $q \in Q$ to $\mathbf{0}$ (respectively $\mathbf{1}$).

To define this operator $\mathcal{Q}$, we will follow the same ideas sketched for the emptiness test. Intuitively, an application of this operator corresponds to one iteration in the computation of the function bad. In the unweighted case, at each of these steps, we performed an inner iteration to compute the auxiliary function adq. Analogously, in order to define the operator $\mathcal{Q}$ we need first to introduce an auxiliary operator $\mathcal{O} : S^Q \to S^Q$. We will focus first on this operator $\mathcal{O}$, which will also be shown to be monotone. The function adq used in the unweighted case actually depends on knowledge of the bad states that have been computed so far; this dependency extends to the weighted case, in order to allow a correct iteration of operator $\mathcal{Q}$ (see page 89). Thus, we actually define one operator $\mathcal{O}_f$ for each $f \in S^Q$. Following the idea of Equation (5.1), the operator $\mathcal{O}_f$ is defined as follows for every $\sigma \in S^Q$ and $q \in Q$:

$$\mathcal{O}_f(\sigma)(q) = \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} \mathsf{wt}(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{k} \mathsf{step}_f(\sigma)(q_j), \tag{5.2}$$

where

$$\mathsf{step}_f(\sigma)(q) = \begin{cases} f(q) & \text{if } q \in F \\ \sigma(q) & \text{otherwise.} \end{cases}$$

In the inner iteration of the emptiness test, the function adq^i is computed by applying Equation 5.1 to a previously computed function adq^i_n until this process stabilizes; that is, until a fixpoint has been found. This iteration is initialized with the function adq^i_0 that maps every state to 1. Since 1 is the least element of the lattice $\mathbb{B}$, the function adq^i_0 is the least element of the lattice S^Q. Thus, the limit of this iteration, i.e., the function adq^i, is in fact the least fixpoint of the operator defined by Equation 5.1 on the lattice S^Q. With the help of the next lemma, we will show that the same idea holds in the operators $\mathcal{O}_f$; that is, that one can compute its least fixpoint by finitely many applications of the operator over the infimum of the lattice S^Q.

Lemma 5.12. *For every $f \in S^Q$ the operator $\mathcal{O}_f$ is monotone, i.e., $\sigma \leq_{S^Q} \sigma'$ implies $\mathcal{O}_f(\sigma) \leq_{S^Q} \mathcal{O}_f(\sigma')$.*

Proof. Let $\sigma, \sigma' \in S^Q$ be such that $\sigma \leq_{S^Q} \sigma'$. This implies also $\mathsf{step}_f(\sigma) \leq_{S^Q} \mathsf{step}_f(\sigma')$. Thus, we have for every $q \in Q$:

$$\begin{aligned} \mathcal{O}_f(\sigma)(q) &= \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} \mathsf{wt}(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \mathsf{step}_f(\sigma)(q_j) \\ &\leq_S \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} \mathsf{wt}(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \mathsf{step}_f(\sigma')(q_j) = \mathcal{O}_f(\sigma'). \end{aligned}$$

□

Since we know that S^Q is finite, this in particular means that the operator $\mathcal{O}_f$ is continuous. By Tarski's fixpoint theorem [Tar55], this implies that the least fixpoint (lfp) of $\mathcal{O}_f$ is $\bigoplus_{n\geq 0} \mathcal{O}_f^n(\widetilde{\mathbf{0}})$. Finiteness of S^Q yields that this lfp is reached after finitely many iterations; more precisely, there exists a smallest $m, 0 \leq m \leq |S|^{|Q|}$ such that $\mathcal{O}_f^m(\widetilde{\mathbf{0}}) = \mathcal{O}_f^{m+1}(\widetilde{\mathbf{0}})$, and for this m we have $\bigoplus_{n\geq 0} \mathcal{O}_f^n(\widetilde{\mathbf{0}}) = \mathcal{O}_f^m(\widetilde{\mathbf{0}})$. This gives us a bound on the number of iterations that is exponential in the size of the automaton. We will later show (see Theorem 5.18) that it is possible to improve this bound to a polynomial number of iterations, measured in the number of states.

Recall our intuition for the auxiliary operator that is trying to find the adequate states. These states are those from which it is possible to construct a finite partial run that finishes in final states that are not known to be bad. In the general case, the operators $\mathcal{O}$ will help in computing the weights of all such runs, which in the end will allow us to help the weights of all successful runs, and hence the behaviour of the automaton. Next, we give a formal definition of the notion of a finite partial run.

Definition 5.13 (Finite run). *A* finite tree *is a finite set $t \subseteq K^*$ that is closed under prefixes and such that, if $ui \in t$ for some $u \in K^*$ and $i \in K$, then for all*

$j, 1 \leq j \leq k, uj \in t$. *A node* $u \in t$ *is called a* leaf *if there is no* $j, 1 \leq j \leq k$ *such that* $uj \in t$. *The set of all leaf nodes of a tree* t *is denoted by* $\mathsf{lnode}(t)$. *The* depth *of a finite tree* t *is the length of the largest word in* t.

A finite run *is a mapping* $r : t \to Q$, *where* t *is a finite tree. Given such a run,* $\mathsf{leaf}(r)$ *denotes the set of all states appearing as labels of a leaf.*

We denote as runs_1 *the set of all runs* r *of depth at least 1 such that for every node* $u \neq \varepsilon$, $r(u) \in F$ *if and only if* u *is a leaf. Additionally,* $\mathsf{runs}_1^{\leq n}$ *denotes the set of all runs in* runs_1 *having depth at most* n. *For a state* $q \in Q$, *we define the sets* $\mathsf{runs}_1(q) = \{r \in \mathsf{runs}_1 \mid r(\varepsilon) = q\}$; *analogously* $\mathsf{runs}_1^{\leq n}(q) = \{r \in \mathsf{runs}_1^{\leq n} \mid r(\varepsilon) = q\}$. *The* weight *of a finite run* $r : t \to Q$ *is* $\mathsf{wt}(r) = \bigotimes_{u \in t \setminus \mathsf{lnode}(t)} \mathsf{wt}(\overrightarrow{r(u)})$. ■

When we are looking for the states that are adequate, we are actually trying to construct a run in runs_1 that starts with each state. Recall from our intuition that we first call adequate any state q having a transition starting with it and leading only to final states. This condition is analogous to having a finite run (of depth 1) in $\mathsf{runs}_1(q)$. We then call adequate any other state p that has a transition leading to adequate or final states; i.e., to non-final states having a run in runs_1 starting with them, or to final states. Concatenating this transition with the runs in runs_1, we obtain a new run in $\mathsf{runs}_1(p)$. This image is nonetheless incomplete, since we are not really interested in any finite run finishing in final states, but only those whose leaf nodes have labels that are not bad. We can see this as multiplying the weight of this run by the function bad applied to each of the states labeling a leaf node. In the general case, consider a given function $f : Q \to S$. We define the *f-weight* of a run r as $\mathsf{wt}_f(r) = \mathsf{wt}(r) \otimes \bigotimes_{q \in \mathsf{leaf}(r)} f(q)$.

We will show that the lfp of the operator $\mathcal{O}_f$ yields the addition of the f-weights of all runs in $\mathsf{runs}_1(q)$ for every state $q \in Q$ with the help of the following lemma.

Lemma 5.14. *For all* $n \geq 0$ *and all* $q \in Q$, $\mathcal{O}_f^n(\widetilde{\mathbf{0}})(q) = \bigoplus_{r \in \mathsf{runs}_1^{\leq n}(q)} \mathsf{wt}_f(r)$.

Proof. The proof is by induction on n. For $n = 0$, the result follows from the fact that $\mathsf{runs}_1^{\leq 0} = \emptyset$, and hence $\bigoplus_{r \in \mathsf{runs}_1^{\leq 0}(q)} \mathsf{wt}_f(r) = \mathbf{0} = \widetilde{\mathbf{0}}(q) = \mathcal{O}_f^n(\widetilde{\mathbf{0}})(q)$.

Assume now that the identity holds for n. Given a tuple $(q_1, \ldots, q_k) \in Q^k$, let $i_1, \ldots, i_l$ be all the indices such that $q_{i_j} \notin F$ for all $j, 1 \leq j \leq l$ and $i_{l+1}, \ldots, i_k$ those indices such that $q_{i_j} \in F$ for all $j, l+1 \leq j \leq k$. Application of the definitions of the operators $\mathcal{O}_f$ and step_f, respectively, yields

$$\begin{aligned}
\mathcal{O}_f^{n+1}(\widetilde{\mathbf{0}})(q) &= \bigoplus_{(q_1,\ldots,q_k) \in Q^k} \mathsf{wt}(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{k} \mathsf{step}_f(\mathcal{O}_f^n(\widetilde{\mathbf{0}}))(q_j) \\
&= \bigoplus_{(q_1,\ldots,q_k) \in Q^k} \mathsf{wt}(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{l} \mathcal{O}_f^n(\widetilde{\mathbf{0}})(q_{i_j}) \otimes \bigotimes_{j=l+1}^{k} f(q_{i_j})
\end{aligned}$$

If $1 \leq j \leq l$, then we will abbreviate $\mathsf{runs}_1^{\leq n}(q_{i_j})$ as rn_j^n and $\mathsf{leaf}(r_j)$ as lf_j. In addition,

we use the symbol F as an abbreviation for the product $\bigotimes_{j=l+1}^{k} f(q_{i_j})$. We then have

$$\mathcal{O}_f^{n+1}(\widetilde{\mathbf{0}})(q) = \bigoplus_{(q_1,\ldots,q_k)\in Q^k} \mathsf{wt}(q,q_1,\ldots,q_k) \otimes (\bigotimes_{j=1}^{l} \bigoplus_{r_j\in\mathsf{rn}_j^n} \mathsf{wt}_f(r_j)) \otimes \mathsf{F} \quad (5.3)$$

$$= \bigoplus_{(q_1,\ldots,q_k)\in Q^k} \mathsf{wt}(q,q_1,\ldots,q_k) \otimes (\bigoplus_{r_1\in\mathsf{rn}_1^n,\ldots,r_l\in\mathsf{rn}_l^n} \bigotimes_{j=1}^{l} \mathsf{wt}_f(r_j)) \otimes \mathsf{F} \quad (5.4)$$

$$= \bigoplus_{(q_1,\ldots,q_k)\in Q^k} \mathsf{wt}(q,q_1,\ldots,q_k) \otimes (\bigoplus_{r_1\in\mathsf{rn}_1^n,\ldots,r_l\in\mathsf{rn}_l^n} \bigotimes_{j=1}^{l} \mathsf{wt}(r_j) \otimes \bigotimes_{p\in\mathsf{lf}_j} f(p)) \otimes \mathsf{F} \quad (5.5)$$

$$= \bigoplus_{(q_1,\ldots,q_k)\in Q^k} \bigoplus_{r_1\in\mathsf{rn}_1^n,\ldots,r_l\in\mathsf{rn}_l^n} \mathsf{wt}(q,q_1,\ldots,q_k) \otimes \bigotimes_{q_j\notin F} \mathsf{wt}(r_j) \otimes \bigotimes_{p\in\mathsf{lf}_j} f(p) \otimes \mathsf{F} \quad (5.6)$$

$$= \bigoplus_{r\in\mathsf{runs}_1^{\leq n+1}(q)} \mathsf{wt}(r) \otimes \bigotimes_{p\in\mathsf{leaf}(r)} f(p) \quad (5.7)$$

$$= \bigoplus_{r\in\mathsf{runs}_1^{\leq n+1}(q)} \mathsf{wt}_f(r).$$

Equation (5.3) applies the induction hypothesis. Identity (5.4) uses the fact that S^Q is a distributive lattice, which allows us to move the addition out of the product, while (5.5) uses the definition of f-weight. Identity (5.6) uses again the distributivity to multiply $\mathsf{wt}(q,q_1,\ldots,q_k)$ inside the addition. Finally, Identity (5.7) simplifies the two sums by constructing a run of larger depth. Instead of considering first the transition $(q,q_1,\ldots,q_k)$ and then runs of depth up to n starting with each q_{i_j}, we simply take the corresponding run of depth $n+1$ starting at q. This run labels the root with q and the successor node i with q_i. If q_i is a final state, then it remains as a leaf, otherwise, below the node i we have the former run starting with q_i. Thus, the set of leafs of this larger run is the union of the sets of leafs of the runs r_js and the set of those q_is that are final states. The last identity merely applies the definition of f-weight again. □

The next theorem shows the relation between the f-weights of the runs in runs_1 and the least fixpoint of the operator $\mathcal{O}_f$.

Theorem 5.15. *Let $f \in S^Q$ and assume that σ_0 is the lfp of the operator $\mathcal{O}_f$. Then, for every $q \in Q$, $\sigma_0(q) = \bigoplus_{r\in\mathsf{runs}_1(q)} \mathsf{wt}_f(r)$.*

Proof. By Lemma 5.14 we know that

$$\bigoplus_{n\geq 0} \mathcal{O}_f^n(\widetilde{\mathbf{0}})(q) = \bigoplus_{n\geq 0} \bigoplus_{r\in\mathsf{runs}_1^{\leq n}(q)} \mathsf{wt}_f(r)$$

$$= \bigoplus_{r\in\mathsf{runs}_1(q)} \mathsf{wt}_f(r).$$

Tarski's fixpoint theorem states that the least fixpoint of $\mathcal{O}_f$ is $\bigoplus_{n\geq 0} \mathcal{O}_f^n(\widetilde{\mathbf{0}})$, which completes the proof of the theorem. □

Before describing how the operators $\mathcal{O}_f$ help in the computation of the behaviour of a weighted automaton, it is worth showing that the number of times it needs to be applied before reaching its lfp is bounded by the number of states of the automaton. The notion of *m-finalising* automata will be useful for this.

Definition 5.16 (m-finalising). *A WBA is* m-finalising *if for every function $f \in S^Q$ and every partial run r in $\mathsf{runs}_1(q)$ there is a partial run s_r in $\mathsf{runs}_1^{\leq m}(q)$ such that $\mathsf{wt}_f(r) \leq_S \mathsf{wt}_f(s_r)$.* ■

We will first show that every WBA is m-finalising for any m grater to the number of non-final states plus one, i.e. $|Q \setminus F| + 1$. Afterwards we will show how this property yields a bound on the number of iterations needed to reach the least fixpoint of $\mathcal{O}_f$.

Theorem 5.17. *Let $\mathcal{A}$ be a WBA with less than $m-1$ non-final states. Then $\mathcal{A}$ is m-finalising.*

Proof. Let $f \in S^Q$ and consider a run $r \in \mathsf{runs}_1(q)$. If $r \in \mathsf{runs}_1^{\leq m}(q)$, then we can consider $s_r = r$, and hence there is nothing to prove.

Otherwise, if $r \notin \mathsf{runs}_1^{\leq m}(q)$, then there must be a path in the tree of length greater than m. As $r \in \mathsf{runs}_1$, in this path there is only one non-root node, namely the leaf node, that is labeled with a final state. Thus, there are at least $m-1$ nodes labeled with non-final states. Since there are less than m different non-final states, there must be two non-root nodes $u \neq v$ in this path such that $r(u) = r(v)$. Since these nodes are in the same path, we can assume w.l.o.g. that $v = uv'$ for some $v' \in K^* \setminus \{\varepsilon\}$. We define a new run s as follows: for every node w if there is no w' for which $w = uw'$, set $s(w) := r(w)$, otherwise (that is, if $w = uw'$ for some w') then set $s(uw') := s(vw')$. This construction defines an injective function g from the nodes of s to the nodes of r such that, for every node w of s, we have $s(w) = r(g(w))$. Notice that this function is *not* surjective, since there is no w such that $g(w) = u$. Thus, s has less nodes than r. Additionally, s is in $\mathsf{runs}_1(q)$. Furthermore, every transition in s is also a transition in r and for every $w \in \mathsf{leaf}(s), g(w) \in \mathsf{leaf}(r)$. This implies that $\mathsf{wt}_f(r) \leq_S \mathsf{wt}_f(s)$. If s is still not in $\mathsf{runs}_1^{\leq m}$, then we can repeat the same process to produce a smaller run s' with a smaller f-weight, until we find one that is in $\mathsf{runs}_1^{\leq m}$. □

We proceed now to show that if we have an m-finalising WBA, then the lfp is found after at most m applications of the operator $\mathcal{O}_f$ to the least element $\widetilde{\mathbf{0}}$. Due to Theorem 5.17, this in particular shows that one needs polynomial time, measured on the number of states of $\mathcal{A}$ to compute this lfp.

Theorem 5.18. *If $\mathcal{A}$ is m-finalising, then $\mathcal{O}_f^m(\widetilde{\mathbf{0}})$ is the lfp of $\mathcal{O}_f$.*

Proof. Let σ_0 be the lfp of $\mathcal{O}_f$. We know that σ_0 is the supremum of $\{\mathcal{O}_f^n(\widetilde{\mathbf{0}}) \mid n \geq 0\}$; thus, it is sufficient to show that $\mathcal{O}_f^m(\widetilde{\mathbf{0}})(q) \geq \sigma_0(q)$ for all $q \in Q$. By Theorem 5.15, we know that $\sigma_0(q) = \bigoplus_{r \in \mathsf{runs}_1(q)} \mathsf{wt}_f(r)$. Since $\mathcal{A}$ is m-finalising, we can replace every

$r \in \mathsf{runs}_1(q)$ by the corresponding $s_r \in \mathsf{runs}_1^{\leq m}(q)$, obtaining a greater element in the lattice. Thus,

$$\begin{aligned}\sigma_0(q) &\leq_S \bigoplus_{r \in \mathsf{runs}_1(q)} \mathsf{wt}_f(s_r) \\ &\leq_S \bigoplus_{s \in \mathsf{runs}_1^{\leq m}(q)} \mathsf{wt}_f(s) = \mathcal{O}_f^m(\widetilde{\mathbf{0}})(q),\end{aligned}$$

which proves our claim. □

The last two theorems tell us that, in order to compute the lfp of an operator $\mathcal{O}_f$, it suffices to apply this operator $|Q \setminus F| + 2$ times. Since each of the iteration steps also requires only polynomial time, measured as a function of the number of states Q, we know that the computation of the lfp needs overall polynomial time in the number of states. This bound is independent of the lattice used. As mentioned before, this bound greatly improves the trivial one obtained from the finiteness of S^Q that is exponential in the number of states of the automaton and also depends on the size of the lattice S.

We focus now on the outer iteration of the algorithm. For this we will define an operator $\mathcal{Q}$ that will allow us to compute the behaviour of $\mathcal{A}$. This operator works in a similar fashion as the iterative computation of all bad states. Recall that in said construction, the set of bad states was updated to include all the states that were detected not to be adequate. In our general case, we have used the operator $\mathcal{O}$ as an analogous of the computation of adequate states. At each step of the outer iteration for computing the function bad, we compute a function adq^i that corresponds to the least fixpoint of the operator from Equation 5.1. This function adq^i was then used to update the knowledge of the bad states. Following the same approach, we define the operator $\mathcal{Q}$ as follows: for all $\sigma \in S^Q$

$$\mathcal{Q}(\sigma) := \mathsf{lfp}(\mathcal{O}_\sigma),$$

where lfp represents the least fixpoint.

We show first that the operator $\mathcal{Q}$ is also monotone and, due to the finiteness of S^Q, its greatest fixpoint can be computed by a repeated application of the operator to the greatest element of the lattice S^Q.

Lemma 5.19. *The operator $\mathcal{Q}$ is monotone.*

Proof. Let $\sigma, \sigma' \in S^Q$ such that $\sigma \leq_{S^Q} \sigma'$. Notice first that, for every run $r \in \mathsf{runs}_1$, this implies that $\mathsf{wt}_\sigma(r) \leq_S \mathsf{wt}_{\sigma'}(r)$. From this we obtain, for every $q \in Q$,

$$\begin{aligned}\mathcal{Q}(\sigma)(q) &= \mathsf{lfp}(\mathcal{O}_\sigma)(q) \\ &= \bigoplus_{r \in \mathsf{runs}_1(q)} \mathsf{wt}_\sigma(r) \qquad (5.8) \\ &\leq_S \bigoplus_{r \in \mathsf{runs}_1(q)} \mathsf{wt}_{\sigma'}(r) \\ &= \mathsf{lfp}(\mathcal{O}_{\sigma'})(q) \qquad (5.9) \\ &= \mathcal{Q}(\sigma'(q),\end{aligned}$$

where Identities (5.8) and (5.9) follow from Theorem 5.15 and the inequality is a consequence of the remark at the beginning of this proof. □

Again, finiteness of S^Q implies that the operator $\mathcal{Q}$ is actually continuous, and thus Tarski's fixpoint theorem says that $\mathcal{Q}$ has $\bigotimes_{n\geq 0} \mathcal{Q}^n(\widetilde{\mathbf{1}})$ as its greatest fixpoint (gfp). It remains to show how this gfp can be used to compute the behaviour of a given WBA. Let $\mathsf{succ}_{\mathcal{A}}(q)$ denote the set of all successful runs of $\mathcal{A}$ whose root is labelled with q. Consider the function $\sigma^{\|} \in S^Q$ where $\sigma^{\|}(q) := \bigoplus_{r\in \mathsf{succ}_{\mathcal{A}}(q)} \mathsf{wt}(r)$. Given this function, we can obtain the behaviour of the WBA $\mathcal{A}$ as follows:

Lemma 5.20. $\|\mathcal{A}\| = \bigoplus_{q\in Q} \mathsf{in}(q) \otimes \sigma^{\|}(q)$.

As it turns out, the function $\sigma^{\|}$ is in fact the greatest fixpoint of $\mathcal{Q}$. In order to prove this claim, we will introduce some additional notation. We will use the expression runs_n, for $n \geq 1$, to denote the set of all finite runs such that every path from the root to a leaf has *exactly* n non-root nodes labeled with a final state, the last of which is the leaf.

Given a run $r \in \mathsf{runs}_n$, its *preamble* is the unique finite run $s \in \mathsf{runs}_1$ such that, for every node u, if $s(u)$ is defined, then $s(u) = r(u)$. We will denote the preamble of r by $\mathsf{pre}(r)$. Notice that if $r \in \mathsf{runs}_n$, for $n \geq 1$, then its preamble always exists, and can be constructed as follows: first set $\mathsf{pre}(r)(\varepsilon) = r(\varepsilon)$ and $\mathsf{pre}(r)(i) = r(i)$ for all $i, 1 \leq i \leq k$. Then, for every node u for which $\mathsf{pre}(r)(u)$ is defined, if $r(u) \in F$, then u is a leaf of $\mathsf{pre}(r)$; otherwise, set $\mathsf{pre}(r)(ui) = r(ui)$ for all $i, 1 \leq i \leq k$. This construction finishes since, in every path, we must find at least one final state, which will become a leaf in $\mathsf{pre}(r)$; thus, it is also the case that $\mathsf{pre}(r) \in \mathsf{runs}_1$.

For a (finite) run r and a node u in r, we will denote the *subrun of r starting at u* as $r_{|u}$. More formally, $r_{|u}$ is the run such that, for every $v \in K^*$, if $r(uv)$ is defined, then $r_{|u}(v) = r(uv)$.

The following lemma relates the number of times n that the operator $\mathcal{Q}$ has been applied to the greatest element $\widetilde{\mathbf{1}}$ of S^Q to the weights of the runs in runs_n.

Lemma 5.21. *For all $n > 0$ and $q \in Q$ it holds that*

$$\mathcal{Q}^n(\widetilde{\mathbf{1}})(q) = \bigoplus_{r\in \mathsf{runs}_n(q)} \mathsf{wt}(r).$$

Proof. We prove this fact also by induction on n. For $n = 1$ the result is a direct consequence of Theorem 5.15. Assume now that it holds for n. From Theorem 5.15 we know that

$$\mathcal{Q}^{n+1}(\widetilde{\mathbf{1}})(q) \;=\; \mathsf{lfp}(\mathcal{O}_{\mathcal{Q}^n(\widetilde{\mathbf{1}})})(q) = \bigoplus_{r\in \mathsf{runs}_1(q)} \mathsf{wt}_{\mathcal{Q}^n(\widetilde{\mathbf{1}})}(r).$$

Using first the definition of f-weights and then the induction hypothesis, we obtain

$$\begin{aligned}\mathcal{Q}^{n+1}(\widetilde{\mathbf{1}})(q) &= \bigoplus_{r\in \mathsf{runs}_1(q)} \mathsf{wt}(r) \otimes \bigotimes_{p\in \mathsf{leaf}(r)} \mathcal{Q}^n(\widetilde{\mathbf{1}})(p) \\ &= \bigoplus_{r\in \mathsf{runs}_1(q)} \mathsf{wt}(r) \otimes \bigotimes_{p\in \mathsf{leaf}(r)} \bigoplus_{s\in \mathsf{runs}_n(p)} \mathsf{wt}(s).\end{aligned}$$

From this equation it then follows that

$$\begin{aligned}
\mathcal{Q}^{n+1}(\widetilde{\mathbf{1}})(q) &= \bigoplus_{r\in\mathsf{runs}_1(q)} \mathsf{wt}(r) \otimes \bigotimes_{u\in\mathsf{lnode}(r)} \bigoplus_{s\in\mathsf{runs}_n(r(u))} \mathsf{wt}(s) && (5.10)\\
&= \bigoplus_{r\in\mathsf{runs}_1(q)} \mathsf{wt}(r) \otimes \bigoplus_{\{t\in\mathsf{runs}_{n+1}(q)\mid\mathsf{pre}(t)=r\}} \bigotimes_{u\in\mathsf{lnode}(r)} \mathsf{wt}(t_{|u}) && (5.11)\\
&= \bigoplus_{r\in\mathsf{runs}_1(q)} \bigoplus_{\{t\in\mathsf{runs}_{n+1}(q)\mid\mathsf{pre}(t)=r\}} \mathsf{wt}(r) \otimes \bigotimes_{u\in\mathsf{lnode}(r)} \mathsf{wt}(t_{|u}) && (5.12)\\
&= \bigoplus_{r\in\mathsf{runs}_1(q)} \bigoplus_{\{t\in\mathsf{runs}_{n+1}(q)\mid\mathsf{pre}(t)=r\}} \mathsf{wt}(t) && (5.13)\\
&= \bigoplus_{s\in\mathsf{runs}_{n+1}(q)} \mathsf{wt}(s). && (5.14)
\end{aligned}$$

Identity (5.10) changes the indices to run over the set of leaf nodes, rather than by the states that label them; the idempotency of the operators $\oplus$ and $\otimes$ implies that this change does not alter the result. For Identity (5.11) we use the distributivity of the lattice. The definition of distributivity says that, in order to exchange the operators $\oplus$ and $\otimes$, the now external addition needs to range over all functions mapping nodes $u \in \mathsf{lnode}(r)$ to runs $s \in \mathsf{runs}_n(r(u))$. We notice that each function of this kind, together with the run $r \in \mathsf{runs}_1(q)$, defines exactly one finite run $t \in \mathsf{runs}_{n+1}(q)$. We thus use this t to represent the function. Identity (5.12) is an easy consequence of distributivity. For Identity (5.13), we then use the fact that a run in runs_{n+1} can be seen as its preamble (in runs_1) concatenated at each of its leafs with a run in runs_n. Finally, for Identity (5.14) we notice that the set of all runs in runs_{n+1} can be partitioned by means of their preambles, which means that both sides of the identity range over the same runs. □

As it was the case for the auxiliary operator $\mathcal{O}$ in the internal iteration, we can bound the number of times that $\mathcal{Q}$ needs to be applied before reaching the greatest fixpoint by the number of states of the automaton. We introduce for this the notion of m-completeness of automata.

Definition 5.22 (m-complete). *A WBA $\mathcal{A}$ is* m-complete *if, for every partial run $r \in \mathsf{runs}_m(q)$, there is a successful run $s_r \in \mathsf{succ}(q)$ such that $\mathsf{wt}(r) \leq_S \mathsf{wt}(s_r)$.* ■

Using the fact that $\otimes$ is idempotent, it is easy to see that every WBA is m-complete for any m greater than the number of final states $|F|$. The proof is similar to the one given in [BHP08] for the fact that a looping automaton has a run iff it has a partial run of depth greater than $|Q|$. However we now need also to take into account which states are final, and which are not.

Theorem 5.23. *Let $\mathcal{A}$ be a WBA with less than m final states; then $\mathcal{A}$ is m-complete.*

Proof. Suppose that we have a partial run $r : t \to Q$ in $\mathsf{runs}_m(q)$. We will use this r to construct a function $\beta : \mathsf{K}^* \to t$ inductively. With this function, we then construct a successful run s_r by setting $s_r(u) := r(\beta(u))$. The intuitive meaning of $\beta(v) = w$ is

that in the run s_r, the node v will have the same label as the node w in r. We define it as follows:

- $\beta(\varepsilon) := \varepsilon$,
- for a node $v \cdot i$, if there is a predecessor w of $\beta(v) \cdot i$ such that (i) $r(\beta(v) \cdot i) = r(w)$, and (ii) $r(w) \in F$, then set $\beta(v \cdot i) := w$; otherwise, set $\beta(v \cdot i) := \beta(v) \cdot i$.

Notice that for every $v \in \mathsf{K}^*$, we have that $\beta(v)$ is not a leaf node of t. In fact, whenever we find a final state twice in the same path, the mapping β leads always to the earliest one. Thus, reaching a leaf would mean that we have a path reaching m final states, where none of them repeats, contradicting the fact that the automaton has less than m final states in total. Hence, the function β is well defined.

We now show that it is possible to construct a successful run s_r from r by defining $s_r(v) = r(\beta(v))$ for all $v \in \mathsf{K}^*$, and that $\mathsf{wt}(r) \leq_S \mathsf{wt}(s_r)$. Our definition of β ensures that, for every $v \in \mathsf{K}^*$ and $i \in \mathsf{K}$ it holds that $s_r(v \cdot i) = r(\beta(v) \cdot i)$. Thus, for every $v \in \mathsf{K}^*$, we have that $(s_r(v), s_r(v1), \ldots, s_r(vk)) = (r(\beta(v)), r(\beta(v) \cdot 1), \ldots, r(\beta(v) \cdot k))$, and hence,

$$\mathsf{wt}(s_r(v), s_r(v1), \ldots, s_r(vk)) = \mathsf{wt}(r(\beta(v)), r(\beta(v) \cdot 1), \ldots, r(\beta(v) \cdot k)).$$

This implies that every factor in the product $\mathsf{wt}(s_r)$ is also a factor in the product $\mathsf{wt}(r)$. Since the product computes the infimum, it holds that $\mathsf{wt}(r) \leq_S \mathsf{wt}(s_r)$.

It remains only to show that s_r is successful. Suppose on the contrary that s_r is not successful. Then, there must exist a path p and a node $v \in \mathsf{p}$ such that all its successors in p are labeled with non-final states. In other words, for every $w \in \mathsf{K}^*$, if $v \cdot w \in \mathsf{p}$, then $s_r(v \cdot w) \notin F$. This implies, by our definition of β, that $\beta(v \cdot w) = \beta(v) \cdot w$, for all $v \cdot w \in \mathsf{p}$. Thus, r has an infinite path, which contradicts the assumption that $r \in \mathsf{runs}_m$. □

The following theorem states that it is possible to compute the mapping $\sigma^{\|}$ for an m-complete automaton by applying the $\mathcal{Q}$ operator to the greatest element $\widetilde{\mathbf{1}}$ of S^Q at most m times.

Theorem 5.24. *If $\mathcal{A}$ is an m-complete WBA, then $\mathcal{Q}^m(\widetilde{\mathbf{1}}) = \sigma^{\|}$.*

Proof. Notice first that by Lemma 5.21, we know that $\mathcal{Q}^m(\widetilde{\mathbf{1}})(q) = \bigoplus_{r \in \mathsf{runs}_m(q)} \mathsf{wt}(r)$. Since $\mathcal{A}$ is m-complete, we can replace each of these partial runs by a successful run, and thus,

$$\begin{aligned} \mathcal{Q}^m(\widetilde{\mathbf{1}})(q) &\leq_S \bigoplus_{r \in \mathsf{runs}_m(q)} \mathsf{wt}(s_r) \\ &\leq_S \bigoplus_{s \in \mathsf{succ}(q)} \mathsf{wt}(s) = \sigma^{\|}(q). \end{aligned}$$

To prove the inequality in the other direction, notice that given a successful run r, we can truncate it at every path when m final states have been found. The result of this is a finite run since otherwise, as the tree is finitely branching, König's Lemma would

imply the existence of an infinite path in this tree. Since branches are truncated once we have found m final states, an infinite path would be one on which less than m final states occur, contradicting the fact that r is a successful run. Thus, the partial run r_m constructed this way belongs to runs_m. Notice that, for every node u of r_m, it holds that $r_m(u) = r(u)$. Hence, we have that $\mathsf{wt}(r) \leq_S \mathsf{wt}(r_m)$. This yields

$$\begin{aligned}\sigma^{\|}(q) = \bigoplus_{r \in \mathsf{succ}(q)} \mathsf{wt}(r) &\leq_S \bigoplus_{r \in \mathsf{succ}(q)} \mathsf{wt}(r_m) \\ &\leq_S \bigoplus_{s \in \mathsf{runs}_m(q)} \mathsf{wt}(s) = \mathcal{Q}^m(\widetilde{\mathbf{1}})(q).\end{aligned}$$

Both inequalities together yield the desired result. □

In particular, this theorem shows that the mapping $\sigma^{\|}$ is indeed the gfp of $\mathcal{Q}$.

Corollary 5.25. *The mapping $\sigma^{\|}$ is the greatest fixpoint of $\mathcal{Q}$.*

Proof. Since S^Q is finite, the gfp of $\mathcal{Q}$ is reached after finitely many iterations; more precisely, if $n_0 > |S|^{|Q|}$, then this gfp is $\bigotimes_{n \geq 0} \mathcal{Q}^n(\widetilde{\mathbf{1}}) = \mathcal{Q}^{n_0}(\widetilde{\mathbf{1}})$. Obviously, we can choose n_0 such that $n_0 > |F|$. Theorem 5.23 then says that the automaton is n_0-complete. Thus, by Theorem 5.24, it follows that $\mathcal{Q}^{n_0}(\widetilde{\mathbf{1}}) = \sigma^{\|}$. □

Overall, we have thus shown how to compute the behaviour of a WBA. By Lemma 5.20, $\|\mathcal{A}\| = \bigoplus_{q \in Q} \mathsf{in}(q) \otimes \sigma^{\|}(q)$. The above corollary says that $\sigma^{\|}$ is the greatest fixpoint of $\mathcal{Q}$, and this fixpoint can be computed in $m_o := |F| + 1$ iteration steps since m_o is larger than the number of final states of the input WBA (Theorems 5.23 and 5.24). Each step of this outer iteration consists of computing the least fixpoint of the operator $\mathcal{O}_\sigma$, where σ is the result of the previous step. This fixpoint can be computed in $m_i = |Q \setminus F| + 2$ iteration steps since m_i is larger than the number of non-final states of the input WBA (Theorems 5.17 and 5.18). Such an inner iteration step requires a polynomial number of lattice operations (in the cardinality $|Q|$ of Q).

Thus, to analyze the *complexity* of our algorithm for computing the behaviour of a WBA, we need to know the complexity of applying the lattice operations. If we assume that this complexity is constant (i.e., the lattice S is assumed to be fixed), then we end up with an overall polynomial time complexity. However, this is not always a reasonable assumption. In fact, we were able to restrict our attention to finite distributive lattices by taking, for a given WBA, the distributive lattice generated by the weights occurring in it (where these weights may come from an underlying infinite distributive lattice). Thus, the actual finite distributive lattice used may depend on the automaton. Let us assume that the lattice operations can be performed using time polynomial in the size of any generating set. Since the size of this generating set is itself polynomial in the number of states of the input WBA $\mathcal{A}$, this assumption implies that the lattice operations can be performed in time polynomial in the size of the automaton. Thus, under this assumption, we have an overall polynomial bound (measured in the number of states) for the computation of the behaviour of a WBA.

In the case of pinpointing, we use the $\mathcal{T}$-Boolean semiring $\mathbb{B}^{\mathcal{T}}$, which is the free distributive lattice generated by the set $\mathsf{lab}(\mathcal{T})$. The lattice operations are conjunction and disjunction of monotone Boolean formulae. Recall that, strictly speaking, the lattice elements are monotone Boolean formulae *modulo equivalence*, i.e., equivalence classes of monotone Boolean formulae. However, since equivalence of monotone Boolean formulae is known to be an NP-complete problem [GJ79], we do not try to compute unique representatives of the equivalence classes. We can instead leave the formulae as they are. Nevertheless, if we are not careful, then the computed pinpointing formula may still be exponential in the size of the automaton, though we apply only a polynomial number of conjunction and disjunction operations. The reason is that we may have to create copies of subformulae. However, this problem can easily be avoided by employing structure sharing, i.e., using directed acyclic graphs (DAGs) as data structure for monotone Boolean formulae. This way, we can compute in polynomial time (a DAG representation of) the pinpointing formula whose size is polynomial in the size of the automaton.[18]

We have now shown that it is possible to compute the behaviour of a WBA in polynomial time measured on the number of states that it has. We have presented two examples of axiomatic automata: a looping automaton for deciding unsatisfiability w.r.t. $\mathcal{SI}$-TBoxes, and a generalised Büchi automaton for deciding axiomatic satisfiability w.r.t. sets of LTL formulae. The pinpointing automata induced by them are thus a WLA and a WGBA, respectively. We will show now that the iterative algorithm for computing the behaviour of WBAs can be used also for computing behaviours of these other two kinds of automata. On one hand, we will see that a WLA is in fact a special case of a WBA, and hence the algorithm works directly. For this special case, though, the method can actually be collapsed to a simpler algorithm where the inner iteration (that is, the computation of the least fixpoint of the operator $\mathcal{O}$) is performed in a trivial step. On the other hand, we will show that for every WGBA we can effectively construct, in polynomial time, a WBA that has the same behaviour, which allows us to reuse the algorithm so far described also in this case.

5.2.2 The Behaviour of WLA

Recall that a WLA is a WGBA that has no set of final states. For a run to be successful in a WGBA, we require that every path in this run has infinitely many nodes labeled with elements of F_i, for each set of final states F_i. In the special case of WLA, this condition is trivially satisfied. Thus, every run of a weighted looping automaton is successful. Alternatively, we can see each WLA $(Q, \mathsf{in}, \mathsf{wt})$ as the WBA $(Q, \mathsf{in}, \mathsf{wt}, Q)$. Forcing every state to be a final state ensures that every run of this automaton is also successful, just as when there were no sets of final states. Thus, the same process for computing the behaviour of WBAs can be applied to WLAs. From Theorem 5.17 we then have that the operators $\mathcal{O}_f$ need to be applied at most twice before reaching its least fixpoint. In fact, in the particular case of WLAs, this bound

[18] Note that the size of the automata we have constructed for $\mathcal{SI}$ and LTL is already exponential in the size of the input. Thus, the pinpointing formula may still be exponential in the size of the input, and computing it may take exponential time in the same measure.

can be further improved to the point where the procedure needs only one iteration, due to a trivialisation of the operator $\mathcal{O}_f$, as we will now show.

Notice first that the operator $\mathcal{O}_f$ depends on the set of final states; more precisely, the function step_f used in the definition of $\mathcal{O}_f$, is divided in two cases, depending on whether the input state is final or not:

$$\mathsf{step}_f(\sigma)(q) = \begin{cases} f(q) & \text{if } q \in F \\ \sigma(q) & \text{otherwise.} \end{cases}$$

If all the states are final, then no case analysis is necessary in step_f, and hence $\mathsf{step}_f(\sigma)(q) = f(q)$ for all $\sigma \in S^Q$ and all $q \in Q$. This collapses the operator $\mathcal{O}_f$ to

$$\mathcal{O}_f(\sigma)(q) = \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} \mathsf{wt}(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{k} f(q_j).$$

Notice that in this case $\mathcal{O}_f$ does not depend on the input σ, and hence its only fixpoint is reached after exactly one iteration. This allows us to accordingly simplify the operator $\mathcal{Q}$ in the following way:

$$\begin{aligned} \mathcal{Q}(\sigma)(q) &= \mathsf{lfp}(\mathcal{O}_\sigma)(q) \\ &= \mathcal{O}_\sigma(\widetilde{\mathbf{0}})(q) \\ &= \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} \mathsf{wt}(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{k} \sigma(q_j). \end{aligned}$$

The behaviour of a WLA is then the gfp of this operator $\mathcal{Q}$, which can be computed by a single iteration without any specific call to $\mathcal{O}_f$. The inner iteration of the procedure for WBAs is replaced in this special case by a direct application of the simplified definition of $\mathcal{Q}$.

Let us apply this insight to the pinpointing automaton for $\mathcal{SI}$ of Definition 5.4. This automaton has exponentially many states in the size n of the input $(C, \mathcal{T})$. Thus, we need exponentially many applications of the operator $\mathcal{Q}$. It is also easy to see that the time required by each application of $\mathcal{Q}$ is exponential in n.

Corollary 5.26. *Let C be an $\mathcal{ALC}$ concept description and $\mathcal{T}$ an $\mathcal{SI}$-TBox. The pinpointing formula for $(C, \mathcal{T})$ w.r.t. unsatisfiability can be computed in time exponential in the size of $(C, \mathcal{T})$.*

Since even deciding satisfiability of $\mathcal{ALC}$ concept descriptions w.r.t. general $\mathcal{SI}$-TBoxes is known to be ExpTime-hard [Sch94], this bound is optimal.

We look now to the more general case of computing the behaviour of WGBAs.

5.2.3 The Behaviour of WGBA

We have shown how to compute the behaviour of a WBA in time polynomial in the number of states. We will now give a polynomial reduction in which, for every

WGBA, we construct a WBA that has the exact same behaviour, reducing in this way the problem of computing the behaviour of WGBAs to the special case of WBAs that we have already solved. For this reduction we once again generalise an idea that has previously been presented for unweighted automata. Intuitively, the reduction consists in creating several copies of the set of states, using one copy to test the Büchi condition for a specific set of final states. When a final state of the current set has been found, we move to the next copy. Between two times that we return to test the first copy, we can be sure that final states from all sets $F_1, \ldots, F_n$ have been found. Thus, it is possible to ensure that the generalised Büchi condition is satisfied. For the unweighted case, this same idea was used to reduce the emptiness problem for GBAs to the one for BAs [VW86]. We formalise now this intuition.

Let $\mathcal{A} = (Q, \mathsf{in}, \mathsf{wt}, F_0, \ldots, F_{n-1})$, with $n > 0$, be a WGBA. We define the WBA $\mathcal{B}_\mathcal{A}$ as the tuple $\mathcal{B}_\mathcal{A} = (Q', \mathsf{in}', \mathsf{wt}', F')$, where

- $Q' = \{(q, i) \mid q \in Q, 0 \leq i \leq n-1\}$,
- $\mathsf{in}'(q, i) = \begin{cases} \mathsf{in}(q) & \text{if } i = 0, \\ \mathbf{0} & \text{otherwise} \end{cases}$
- $\mathsf{wt}'((q_0, i), (q_1, j), \ldots, (q_k, j)) = \begin{cases} \mathsf{wt}(q_0, q_1, \ldots, q_k) & \text{if } q_0 \in F_i,\ j = i + 1 \bmod n, \\ \mathsf{wt}(q_0, q_1, \ldots, q_k) & \text{if } q_0 \notin F_i,\ i = j \\ \mathbf{0} & \text{otherwise} \end{cases}$
- $F' = \{(q, n-1) \mid q \in F_{n-1}\}$.

Notice that the automaton $\mathcal{B}_\mathcal{A}$ has $n \cdot |Q|$ states, where n is the number of sets of final states in $\mathcal{A}$. Since there can potentially be $2^{|Q|}$ sets of final states, this reduction is not polynomial when measured only in the number of states in $\mathcal{A}$, but it is still polynomial in the total size of the original automaton $\mathcal{A}$.

Definition 5.27 (Support). *Let $\mathcal{A}$ be a WGBA. The* support of $\mathcal{A}$, *denoted as* $\mathsf{supp}(\mathcal{A})$, *is the set of all runs r such that* $\mathsf{in}(r(\varepsilon)) \otimes \mathsf{wt}(r) \neq \mathbf{0}$. ■

The behaviour of a weighted automaton is, by definition, the supremum (that is, the addition) of the weights of all successful runs multiplied by the initial distribution of their root labels. Obviously, if a run r is such that $\mathsf{in}(r(\varepsilon)) \otimes \mathsf{wt}(r) = \mathbf{0}$, i.e., if $r \notin \mathsf{supp}(\mathcal{A})$, then it will not have any influence in the computed behaviour, and can hence be ignored. Our proof of behaviour-equivalence of $\mathcal{A}$ and $\mathcal{B}_\mathcal{A}$ will show that there is a bijection between their supports that is weight preserving.

Theorem 5.28. *If $\mathcal{A}$ is a WGBA with at least one set of final states and $\mathcal{B}_\mathcal{A}$ is constructed as above, then $\|\mathcal{A}\| = \|\mathcal{B}_\mathcal{A}\|$.*

Proof. We will introduce a bijective function $f : \mathsf{supp}(\mathcal{A}) \to \mathsf{supp}(\mathcal{B}_\mathcal{A})$ such that, for every run $r \in \mathsf{supp}(\mathcal{A})$, it holds that (i) $\mathsf{wt}(r) = \mathsf{wt}'(f(r))$ and (ii) r is successful (w.r.t. $\mathcal{A}$) iff $f(r)$ is successful (w.r.t. $\mathcal{B}_\mathcal{A}$).

Let r be a run in $\mathsf{supp}(\mathcal{A})$. We define the run $f(r)$ of $\mathcal{B}_\mathcal{A}$ recursively as follows:

- $f(r)(\varepsilon) = (r(\varepsilon), 0)$;

- let $u \in \mathsf{K}^*$ and $f(r)(u) = (q, i)$. Then, for all $1 \leq j \leq k$,

$$f(r)(uj) = \begin{cases} (r(uj), i) & \text{if } q \notin F_i, \\ (r(uj), i + 1 \bmod n) & \text{if } q \in F_i. \end{cases}$$

Let $u \in \mathsf{K}^*$, and $f(r)(u) = (q, i)$. Then $r(u) = q$. Furthermore, for all $1 \leq j \leq k$, it holds that $f(r)(uj) = (r(uj), i+1 \bmod n)$ if $q \in F_i$ and $f(r)(uj) = (r(uj), i)$ otherwise. Together with the definition of wt', this implies

$$\mathsf{wt}'(f(r)(u), f(r)(u1), \ldots, f(r)(uk)) = \mathsf{wt}(r(u), r(u1), \ldots, r(uk)).$$

And thus, we have that $\mathsf{wt}(r) = \mathsf{wt}'(f(r))$. Since we also have $\mathsf{in}'(f(r)(\varepsilon)) = \mathsf{in}(r(\varepsilon))$, the fact that $\mathsf{in}(r(\varepsilon)) \otimes \mathsf{wt}(r) \neq \mathbf{0}$ also implies that $\mathsf{in}'(f(r)(\varepsilon)) \otimes \mathsf{wt}'(f(r)) \neq \mathbf{0}$. This means that f is indeed a function from $\mathsf{supp}(\mathcal{A})$ to $\mathsf{supp}(\mathcal{B}_\mathcal{A})$.

It is easy to see that f is injective. We show now that it is also surjective. Consider a run $s \in \mathsf{supp}(\mathcal{B}_\mathcal{A})$. We need to show that there exists a run $r \in \mathsf{supp}(\mathcal{A})$ such that $s = f(r)$. We construct the run $r \in \mathsf{supp}(\mathcal{A})$ as follows: for every $u \in \mathsf{K}^*$, if $s(u) = (q, i)$, then $r(u) = q$. We show now that $s = f(r)$. First, since $s \in \mathsf{supp}(\mathcal{B}_\mathcal{A})$, it holds that $\mathsf{in}'(s(\varepsilon)) \otimes \mathsf{wt}'(s) \neq \mathbf{0}$. This in particular means that $\mathsf{in}'(s(\varepsilon)) \neq \mathbf{0}$, and thus, $s(\varepsilon) = (q, 0)$ for some $q \in Q$. Consider now a $u \in \mathsf{K}^*$ and let $s(u) = (q, i)$. Hence, also $r(u) = q$. Since $\mathsf{wt}'(s(u), s(u1), \ldots, s(uk)) \neq \mathbf{0}$, it must be the case that for all $j, 1 \leq j \leq k$ it holds that, if $q \notin F_i$, then $s(uj) = (q_j, i)$, and if $q \in F_i$, then $s(uj) = (q_j, i + 1 \bmod n)$, for some $q_j \in Q$. But then, s satisfies the definition of $f(r)$, which shows that f is surjective.

It remains only to show that r is successful (w.r.t. the WGBA $\mathcal{A}$) iff $f(r)$ is successful (w.r.t. the WBA $\mathcal{B}_\mathcal{A}$). Suppose first that $f(r)$ is successful. Then for every path there are infinitely many nodes labeled with elements of the only set of final states $F' = \{(q, n-1) \mid q \in F_{n-1}\}$. But notice that, according to the way f was defined, if $f(r)(u) \in F'$, then $f(r)(uj)$ is of the form $(q_j, 0)$ for all $1 \leq j \leq k$. All the following nodes in the path will have labels of the form $(\_, 0)$ until a state from F_0 is found; at which point, the labels will be changed to the form $(\_, 1)$, and so on. Thus, for each u such that $f(r)(u) \in F'$ there exist $v_0, v_1, \ldots, v_{n-1}$ such that for every $i, 0 \leq i < n$, there is a $q_i \in F_i$ with $f(r)(u \cdot v_0 \cdots v_i) = (q_i, i)$, and hence $r(u \cdot v_0 \cdots v_i) = q_i \in F_i$. This implies that r is successful.

Conversely, assume that $f(r)$ is not successful. Then, there is a path $u_1, u_2, \ldots$ and a $l \geq 0$ such that for all $l' \geq l$ it holds that $f(r)(u_{l'}) \notin F'$. Since the second component can only increase (modulo n) from a node in a path to the other, there must be a $1 \leq i_0 \leq n$ such that $f(r)(u_{l'})$ is of the form $(q_{l'}, i_0)$ for all $l' \geq l$. But this means that for all $l' \geq l$, $r(u_{l'}) \notin F_{i_0}$. Thus, r is also not a successful run.

From this bijection between the runs in the supports, the equivalence in the be-

haviours can be deduced as follows.

$$\begin{aligned}
\|\mathcal{A}\| &= \bigoplus_{r \text{ successful run of } \mathcal{A}} \mathsf{in}(r(\varepsilon)) \otimes \mathsf{wt}(r) \\
&= \bigoplus_{r \text{ successful run of } \mathcal{A}} \mathsf{in}(r(\varepsilon)) \otimes \mathsf{wt}(f(r)) \\
&= \bigoplus_{f(r) \text{ successful run of } \mathcal{B}_{\mathcal{A}}} \mathsf{in}(f(r)(\varepsilon)) \otimes \mathsf{wt}(f(r)) \\
&= \bigoplus_{r \text{ successful run of } \mathcal{B}_{\mathcal{A}}} \mathsf{in}'(r(\varepsilon)) \otimes \mathsf{wt}'(r) = \|\mathcal{B}_{\mathcal{A}}\|,
\end{aligned}$$

which concludes our proof. □

Given a WGBA with m states and n sets of final states, this reduction yields a WBA with $n \cdot m$ states. As described before, computing the behaviour of a WBA requires time polynomial in the size of its state set; in this case, polynomial in $n \cdot m$. Thus, our method computes the behaviour of a WGBA in time polynomial in the overall number of states and sets of final states that it contains.

Let us apply this approach for computing the behaviour of a WGBA to the pinpointing automaton for LTL from Definition 5.6. This automaton has exponentially many states in the size n of the input $(\phi, \mathcal{R})$ and linearly many set of final states in n. Thus, the WBA constructed from the WGBA is of size exponential in n. Overall, the two nested iterations perform exponentially many steps, which leads to an algorithm with a total running time that is exponential in the size of the input.

Corollary 5.29. *Let ϕ be an LTL formula and $\mathcal{R}$ a set of LTL formulae. A pinpointing formula for $(\phi, \mathcal{R})$ w.r.t. a-unsatisfiability can be computed in time exponential in the size of $(\phi, \mathcal{R})$.*

5.3 An Alternative Computation of the Behaviour

Independently from the development of the present dissertation, a different algorithm for computing the behaviour of WBAs over distributive lattices was developed by Droste *et al.* [DKR08]. We will first sketch this alternative approach and then compare it to ours, with special attention to the application in the pinpointing scenario.[19] In the following, we will call our method the *iterative method* and the one from [DKR08] the *prime method*.

The prime method is based on the following property of distributive lattices. Let $(S, \leq_S)$ be a distributive lattice. An element $p \in S$ is called *meet prime* if, for every $s_1, s_2 \in S$, $s_1 \otimes s_2 \leq_S p$ implies that either $s_1 \leq_S p$ or $s_2 \leq_S p$. It is known that

[19] We present only a special case of the algorithm in [DKR08], where we allow only unlabeled trees as inputs. Furthermore, we have exchanged the use of *join prime* elements in [DKR08] with the use of their *meet prime* counterparts. This is justified by duality of distributive lattices, allows for an easier understanding of how this method works in the pinpointing application, and makes it easier to compare it with our approach in this setting.

any element s of S equals the infimum of all the meet prime elements greater than or equal to s [Grä98]. If one could decide, for a given meet prime element p, whether p is greater than or equal to the behaviour of a weighted automaton, then this behaviour could be readily found from the outputs of such decisions by computing the infimum of all those meet prime elements for which this decision is answered positively.

In the prime method, this decision problem is solved in the following way. Let $\mathcal{A} = (Q, \mathsf{in}, \mathsf{wt}, F)$ be the WBA over the distributive lattice $(S, \leq_S)$ for which we want to compute the behaviour, and let $\mathsf{prime}(S)$ denote the set of all meet prime elements of S. For every meet prime element $p \in \mathsf{prime}(S)$, construct the (unweighted) Büchi automaton $\mathcal{A}_p = (Q, \Delta, I, F)$ where:

- $\Delta := \{(q, q_1, \ldots, q_k) \in Q^{k+1} \mid \mathsf{wt}(q, q_1, \ldots, q_k) \not\leq_S p\}$;
- $I := \{q \in Q \mid \mathsf{in}(q) \not\leq_S p\}$.

It is easy to see that $\mathcal{A}_p$ accepts a non-empty language, i.e., there exists a successful run of $\mathcal{A}_p$ that starts with an initial state, iff there is a successful run r of $\mathcal{A}$ such that $\mathsf{in}(r(\varepsilon)) \otimes \mathsf{wt}(r) \not\leq_S p$. Equivalently, the language accepted by $\mathcal{A}_p$ is empty iff, for every successful run r of $\mathcal{A}$, it holds that $\mathsf{in}(r(\varepsilon)) \otimes \mathsf{wt}(r) \leq_S p$. But this means that $\|\mathcal{A}\| \leq_S p$. Thus, if we denote by $\mathcal{L}(\mathcal{A}_p)$ the language accepted by the automaton $\mathcal{A}_p$, we have

$$\|\mathcal{A}\| = \bigotimes_{\{p \in \mathsf{prime}(S) \mid \mathcal{L}(\mathcal{A}_p) = \emptyset\}} p.$$

In the pinpointing application, we use the lattice $\mathbb{B}^{\mathcal{T}}$, where the meet prime elements are exactly all conjunctions of propositional variables in $\mathsf{lab}(\mathcal{T})$.[20] There is then a one-to-one correspondence between the meet prime elements of $\mathbb{B}^{\mathcal{T}}$ and all subsets of axioms appearing in the axiomatic input for which the pinpointing formula is being computed. Take an arbitrary meet prime element p and assume that it corresponds to the set of axioms $\mathcal{T}' \subseteq \mathcal{T}$, i.e., $p = \bigwedge_{t \in \mathcal{T}'} \mathsf{lab}(t)$. The automaton $\mathcal{A}_p$ has a transition $(q, q_1, \ldots, q_k)$ iff

$$\Delta\mathsf{vio}(q, q_1, \ldots, q_k) = \mathsf{wt}(q, q_1, \ldots, q_k) \not\leq_{\mathbb{B}^{\mathcal{T}}} p = \bigwedge_{t \in \mathcal{T}'} \mathsf{lab}(t).$$

Since $\Delta\mathsf{vio}(q, q_1, \ldots, q_k) = \bigvee_{\{t \in \mathcal{T} \mid (q, q_1, \ldots, q_k) \notin \Delta\mathsf{res}(t)\}} \mathsf{lab}(t)$, this means that for every $t \in \mathcal{T}'$, $(q, q_1, \ldots, q_k) \in \Delta\mathsf{res}(t)$. But this holds iff $(q, q_1, \ldots, q_k)$ is a transition of $\mathcal{A}_{|\mathcal{T}'}$ (see Definition 5.1). Analogously, it is easy to see that a state q is an initial state of $\mathcal{A}_p$ iff it is an initial state of $\mathcal{A}_{|\mathcal{T}'}$. Thus, the automaton $\mathcal{A}_p$ is identical to the $\mathcal{T}'$-restricted subautomaton $\mathcal{A}_{|\mathcal{T}'}$. Consequently, testing the automaton $\mathcal{A}_p$ for emptiness is the same as testing $\mathcal{A}_{|\mathcal{T}'}$ for emptiness. This shows that the prime method actually corresponds to the naïve black-box approach of testing the c-property for all possible subsets of axioms. Unoptimized, this process will thus always need an exponential number of tests for computing the pinpointing formula. However, this process allows

[20] Recall that the lattice $\mathbb{B}^{\mathcal{T}}$ uses disjunction as its infimum operator, and conjunction as the supremum. Thus, conjunctions of variables are the only elements of the lattice that cannot be written as the infimum (disjunction) of other elements.

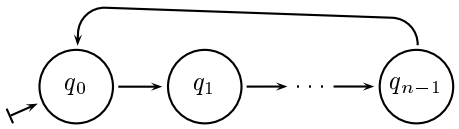

Figure 5.1: The looping automaton $\mathcal{A}_n$ from Example 5.30.

the use of all the optimizations applicable to black-box pinpointing algorithms, which are independent of the procedure used to decide the underlying property. Notice, nonetheless, that finding all prime elements that are greater than or equal to the behaviour is equivalent to finding all sets of axioms that contain at least one MinA. As a consequence of this, there are cases where an exponential number of emptiness tests is necessary, even when using black-box optimizations (see Chapter 6).

In the examples we have presented in this work (i.e., pinpointing unsatisfiability in $\mathcal{SI}$ and LTL), both the iterative and the prime method have an exponential running time. For the iterative method, we have a bound that is polynomial in the number of states of the constructed automata, but this number is itself exponential in the size of the input. The prime method performs exponentially many emptiness tests, each of which requires exponential time (since it is performed on an exponentially large automaton). Although both approaches result in an exponential-time algorithm in these cases, the bound on the iterative method has the advantage of not depending on the number of meet prime elements of the lattice, as opposed to the prime method. In the case of pinpointing, the lattice has always 2^n meet prime elements, where n is the number of input axioms. If the axiomatic automaton deciding the property has a number of states polynomial in the size of the input, then this exponential number of tests will yield a suboptimal procedure, as demonstrated by the following examples.

Example 5.30. *Consider an input $\mathcal{I}$ and a set of axioms $\mathcal{T} = \{t_0, \ldots, t_{n-1}\}$, and assume that the c-property is defined as follows: $\mathcal{P}_\mathbf{1} := \{(\mathcal{I}, \mathcal{T}') \mid \mathcal{T}' \subseteq \mathcal{T}, |\mathcal{T}'| > 0\}$. Let each axiom t_i be labelled with the propositional variable p_i. Then a pinpointing formula for $\mathcal{P}_\mathbf{1}$ is given by $\bigvee_{0 \leq i < n} p_i$.*

We can construct an axiomatic automaton $(\mathcal{A}_n, \Delta\mathsf{res}, I\mathsf{res})$ for the axiomatised input $(\mathcal{I}, \mathcal{T})$ as follows:

- *$\mathcal{A}_n$ is the looping automaton for arity 1 $\mathcal{A}_n := (\{q_0, \ldots, q_{n-1}\}, \Delta, \{q_0\})$ depicted in Figure 5.1, where*
- *$\Delta = \{(q_i, q_{(i+1) \bmod n}) \mid 0 \leq i < n\}$;*
- *for every $0 \leq j \leq n-1$, $\Delta\mathsf{res}(t_j) = \Delta \setminus \{(q_j, q_{(j+1) \bmod n})\}$;*
- *for every $t \in \mathcal{T}$, $I\mathsf{res}(t) = \{q_0\}$.*

It is easy to see that this axiomatic automaton is correct for the property $\mathcal{P}_\mathbf{1}$. Since $\mathcal{A}_n$ has n states and n transitions, the iterative method needs polynomial time to compute the behaviour of the pinpointing automaton induced by $(\mathcal{A}_n, \Delta\mathsf{res}, I\mathsf{res})$, measured in the number of axioms n. On the other hand, the unoptimized prime method requires 2^n emptiness tests. ■

We will take advantage of this example to illustrate how the iterative method computes the behaviour of an automaton (which in this case corresponds to the pinpointing formula). The axiomatic automaton $(\mathcal{A}_n, \Delta\mathsf{res}, I\mathsf{res})$ induces the pinpointing automaton $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin} = (\{q_0, \ldots, q_{n-1}\}, \mathsf{in}, \mathsf{wt})$, where

- $\mathsf{in}(q_0) = \bot$ and $\mathsf{in}(q_i) = \top$ for all $0 < i < n$; and
- $\mathsf{wt}(q_i, q_j)$ equals p_i if $j = (i+1) \bmod n$, and $\top$ otherwise.

As this is a weighted looping automaton, the iterative method reduces to an iterated application of the simplified operator $\mathcal{Q}$ described in Section 5.2.2. Notice that, for every state q_i, there is exactly one transition, namely $(q_i, q_{(i+1) \bmod n})$, having a weight distinct from $\top$. Hence, for every function $\sigma : Q \to \mathbb{B}^{\mathcal{T}}$ we have:

$$\begin{aligned} \mathcal{Q}(\sigma)(q_i) &= \bigwedge_{0 \le j < n} \mathsf{wt}(q_i, q_j) \vee \sigma(q_j) \\ &= \mathsf{wt}(q_i, q_{(i+1) \bmod n}) \vee \sigma(q_{(i+1) \bmod n}) = p_i \vee \sigma(q_{(i+1) \bmod n}). \end{aligned}$$

The process starts with the function $\widetilde{\mathbf{1}} : Q \to \mathbb{B}^{\mathcal{T}}$ that maps every state to $\bot$; that is, $\widetilde{\mathbf{1}}(q_i) = \bot$ for all $0 \le i < n$. After the first application of the operator $\mathcal{Q}$, we have $\mathcal{Q}(\widetilde{\mathbf{1}})(q_i) = p_i$ for all $0 \le i < n$ since $p_i \vee \bot$ is equivalent to p_i. Analogously, after m iterations we have, for all $0 \le i < n$, that

$$\mathcal{Q}^m(\widetilde{\mathbf{1}})(q_i) = \bigvee_{0 \le j < m} p_{(i+j) \bmod n}.$$

This process reaches a fixpoint when $m = n$, in which case every state q_i is mapped to the formula $\bigvee_{0 \le j < n} p_j$. Thus, the behaviour of $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}$ is

$$\begin{aligned} \|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}\| &= \textstyle\bigwedge_{0 \le i < n} \mathsf{in}(q_i) \vee \mathcal{Q}^n(\widetilde{\mathbf{1}})(q_i) \\ &= \mathsf{in}(q_0) \vee \mathcal{Q}^n(\widetilde{\mathbf{1}})(q_0) \\ &= \mathcal{Q}^n(\widetilde{\mathbf{1}})(q_0) = \textstyle\bigvee_{0 \le j < n} p_j, \end{aligned}$$

which is a pinpointing formula.

We present a second example in which the original decision procedure requires a generalised Büchi acceptance condition. This additional example shows that the exponential blowup in the execution time of the prime method when compared to the iterative method can appear also with properties for which the looping acceptance condition is not sufficient.

Example 5.31. *Let* Q *be an infinite set of states and let the set of inputs* $\mathfrak{I}$ *be the set of all generalised Büchi automata using states from* Q*, and the set of axioms be* $\mathfrak{T} := \mathsf{Q}^{k+1}$*. That is, we use the transitions of the automata in* $\mathfrak{I}$ *as axioms of our property. We define the c-property* $\mathcal{P}_\mathbf{2}$ *as the set of all tuples of the form* $(\mathcal{A}, \Theta)$ *where* $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$ *is a generalised Büchi automaton in* $\mathfrak{I}$*, and* $\Theta \subseteq \mathfrak{T}$*, such that* $(Q, \Delta \setminus \Theta, I, F_1, \ldots, F_n)$ *has no successful run* r *with* $r(\varepsilon) \in I$*. Intuitively, the axioms tell which transitions are* disallowed *in the input automaton* $\mathcal{A}$*. The c-property*

is satisfied whenever we remove enough transitions (by adding them to the axiom set) to avoid any successful run whose root is labelled with an initial state. It is easy to see that the axiomatic automaton $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ *where* $\Delta\mathsf{res}(t) = \Delta \setminus \{t\}$ *and* $I\mathsf{res}(t) = Q$ *for all* $t \in \Theta$ *is correct for the property* $\mathcal{P}$ *and the axiomatised input* $(\mathcal{A}, \Theta)$*. As we have seen, the iterative method requires time polynomial in the number of states* $|Q|$ *of this axiomatic automaton to compute the pinpointing formula for this property. On the other hand, the prime method needs* $2^{|\Theta|}$ *emptiness tests, each polynomial in* $|Q|$*. We thus have a potentially exponential increase in execution time, when compared to the iterative method.* ■

One advantage of the prime method is that it can easily be generalised to more complex automata models. For instance, it is shown in [DKR08] how the same idea works in the presence of a more complex acceptance condition, known as the Muller condition. Also note that the prime method can possibly be optimized using the ideas underlying the known optimizations of black-box pinpointing procedures, not just in the case of applying it to pinpointing, but also in a more general setting.

In this chapter we have introduced a general method for computing the pinpointing formula of any c-property that can be decided with an axiomatic automaton using a Büchi acceptance condition. We do this through the construction of the pinpointing automaton induced by the original axiomatic automaton. The pinpointing automaton is a weighted automaton whose behaviour is a pinpointing formula. In order to effectively compute the formula, we developed an algorithm that computes the behaviour of weighted automata over finite distributive lattices. This method generalises the ideas employed for the well-known iterative emptiness test on unweighted automata. We also described how this iterative method can be used, along with an adequate data structure, to construct the pinpointing formula in time polynomial in the size of the automaton. Since just deciding the emptiness of automata in general requires polynomial time in the same measure, the iterative algorithm turns out to be optimal from a complexity point of view.

We instantiated our approach by showing how it can be used to compute a pinpointing formula for unsatisfiability of $\mathcal{ALC}$ concept terms w.r.t. general $\mathcal{SI}$-TBoxes, as well as for axiomatic unsatisfiability of LTL formulae. In both cases, the automaton constructed has size exponential in the number of axioms, and thus the algorithm requires exponential time to compute the pinpointing formula. This bound is optimal for unsatisfiability of $\mathcal{ALC}$ concept terms w.r.t. general $\mathcal{SI}$-TBoxes, where the underlying decision problem is already ExpTime hard [Sch94]. On the other hand, deciding axiomatic unsatisfiability of LTL formulae is in PSpace [SC85], and it is unclear whether the automata-based decision procedure yields an optimal time bound or not.

In the next chapter we will look in detail at some complexity results for pinpointing. Although the focus on this work has been on computing a pinpointing formula, due to the fact that all MinAs and MaNAs can then be deduced from it, our complexity study will primarily look at the hardness of finding these sets of axioms, rather than the mentioned formula.

Chapter 6

Complexity Results

So far in this work we have focused on *how* to compute a pinpointing formula for a given property $\mathcal{P}$ by extending the procedure used for deciding $\mathcal{P}$. For the pinpointing extension of general tableaux, we found a problem even for ensuring a finite execution time. We had to settle for a subclass of tableaux, claiming that it is impossible to fully characterize the set of all tableaux having a terminating pinpointing extension. Even in the cases of termination, it is not clear how the labeling mechanism used in the pinpointing extension affects the overall execution time. If we restrict the discourse to *ground* tableaux (see Definition 3.5), then we know that the pinpointing extension will generate the same set of assertions as the original tableau algorithm, but may change their labels exponentially often, in the number of axioms, as there are exponentially many monotone Boolean formulae that can label each assertion. Thus, the pinpointing extension of ground tableaux has an execution time exponential in the number of axioms. This in particular means that the pinpointing extension of the tableau for subsumption of $\mathcal{HL}$ concept names requires exponential time, although the underlying decision procedure terminates in polynomial time in the number of axioms.

For the case of automata-based decision procedures, we showed that the pinpointing formula can be computed in time polynomial in the size of the automaton. Since merely deciding the property requires time polynomial in the same measure, this method is optimal with respect to its underlying decision procedure. In other words, if the axiomatic automaton $\mathcal{A}$ is an optimal decision procedure for the property $\mathcal{P}$, then the pinpointing automaton induced by $\mathcal{A}$ computes the pinpointing formula in optimal time. For instance, unsatisfiability of $\mathcal{ALC}$ concept terms w.r.t. general $\mathcal{SI}$-TBoxes is an ExpTime complete problem, and the axiomatic automaton $(\mathcal{A}_{C,\mathcal{T}}, \Delta\mathsf{res}_{C,\mathcal{T}}, I\mathsf{res}_{C,\mathcal{T}})$ that decides this property has size exponential in the number of axioms. Thus, a pinpointing formula can be computed from its pinpointing automaton in exponential time. But it might well be the case that the automaton used yields a suboptimal decision procedure. For instance, the axiomatic automaton $(\mathcal{A}_{\phi,\mathcal{R}}, \Delta\mathsf{res}_{\phi,\mathcal{R}}, I\mathsf{res}_{\phi,\mathcal{R}})$ has also size exponential in the number of axioms, but the property it decides, namely axiomatic unsatisfiability of LTL formulae, is known to be in PSpace [SC85]. Using the pinpointing automaton to compute the pinpointing formula yields an exponential time algorithm. It is unclear whether this algorithm is

optimal or not.

In this chapter we try to shine some light on the hardness of solving pinpointing-related problems. We divide this study into two parts. First, we show complexity results that are independent of the method use for solving the problems. Afterwards, we prove our claim from Chapter 3 that it is undecidable whether the pinpointing extension of a terminating general tableau is also terminating.

6.1 Complexity of Pinpointing

We start our study of the complexity of pinpointing by showing a trivial upper bound obtained by the simplest black-box algorithm. Let $\mathcal{P}$ be a c-property and $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatised input such that $\Gamma \in \mathcal{P}$. Given an arbitrary procedure that decides $\mathcal{P}$, we can find the set of all MinAs, all MaNAs and a pinpointing formula for $\mathcal{P}$ and Γ, with a very naïve black-box algorithm that consists on applying the decision procedure $2^{|\mathcal{T}|}$ times. One simply tests, for each $\mathcal{T}' \subseteq \mathcal{T}$, whether $(\mathcal{I}, \mathcal{T}') \in \mathcal{P}$ or not. From the answers to these tests, the sets $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ and $\mathsf{MAX}_{\mathcal{P}(\Gamma)}$ can readily be computed, and hence also the pinpointing formula (see Page 37). This in particular means that, if the decision procedure runs in *at most* exponential time, then $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$, $\mathsf{MAX}_{\mathcal{P}(\Gamma)}$ and the pinpointing formula can be computed in exponential time.[21] Obviously, for any c-property whose decision problem is ExpTime-complete, such as unsatisfiability of $\mathcal{ALC}$ concept terms w.r.t. general TBoxes [Sch91, BCM+03], this bound is tight. We will see that even for problems in lower complexity classes, the bound is also tight. Along with this, we will analyse the complexity of other problems related to pinpointing.

As we want to identify how much of the complexity is due to pinpointing, as opposed to the original decision problem, our results will be based on subsumption of $\mathcal{HL}$ concept names. Since this property is decidable in polynomial time, any increase in complexity that we encounter can then be attributed to pinpointing.

This section is composed of three parts. In the first part we present complexity results related to the computation of MinAs. Some of these results first appeared in [BPS07a], where it was also claimed, without proof, that their dual results hold also for the computation of MaNAs. In the second part we present proofs to this claim. Finally, in Section 6.1.3, we show that there exist axiomatised inputs for which the pinpointing formula has superpolynomial length, when measured in the number of axioms. This in particular implies that such a formula cannot be written (nor computed) in polynomial time.

6.1.1 MinA Complexity

If we are only interested in finding one, arbitrary, MinA, then we can compute it with a black-box algorithm that calls the decision procedure only $|\mathcal{T}|$ times [BPS07a, Chi97, KPSG06]. The idea consists in systematically trying to remove axioms while still

[21] Notice that this also implies that if the decision procedure is *at least* exponential, then pinpointing-related problems are solvable without an increment in the complexity.

belonging to the property. Suppose that we have some $\mathcal{T}' \subseteq \mathcal{T}$ such that $(\mathcal{I}, \mathcal{T}') \in \mathcal{P}$. We then select an axiom $t \in \mathcal{T}'$ that is going to be tested for removal. If the property still follows once t is removed, i.e., if $(\mathcal{I}, \mathcal{T}' \setminus \{t\}) \in \mathcal{P}$, then t is not necessary for the property to hold and hence can be removed. This process is then repeated with $\mathcal{T}' \setminus \{t\}$. If, on the contrary, $(\mathcal{I}, \mathcal{T}' \setminus \{t\}) \notin \mathcal{P}$, then we know that t must belong to all MinAs for $\mathcal{T}'$, and we hence continue the process with the set $\mathcal{T}'$, but never testing t for removal again. In this way, every axiom is tested for removal exactly once. It can be shown that the set of axioms resulting from this procedure is indeed a MinA. Thus, the computation of one arbitrary MinA is in the same complexity class as merely deciding the underlying property, as long as this latter problem is at least polynomial. In the case of subsumption of $\mathcal{HL}$ concept names, this means that one MinA can be computed in polynomial time in the size of the TBox.

If we further assume that the axioms in the TBox are ordered, then we can find the *lexicographical last* MinA also in polynomial time. We say that a set of axioms S is *lexicographically before* another set S' iff the first element at which they disagree is in S. If we test the axioms for removal in order, then the black-box algorithm described above yields the last lexicographical last MinA.[22] Also the additive algorithm by Tamiz, Mardle and Jones [TMJ96] (see also [Chi97]) yields as an output the lexicographical last MinA in polynomial time.

Unfortunately, computing one MinA, even the lexicographical last one, is usually not enough. For instance, if we are trying to understand why an axiomatic input belongs to a c-property, then it would be desirable to obtain MinAs that have as few axioms as possible, as larger sets of axioms are more difficult to interpret. The following theorem shows that deciding the existence of a MinA whose cardinality is bounded by a given natural number n is an NP-complete problem (see [Sun09, BPS07a] for a proof). Hence, it is hard to know whether a given MinA has minimal size or not.

Theorem 6.1. *Given an $\mathcal{HL}$ TBox $\mathcal{T}$, concept names A, B occurring in $\mathcal{T}$, and a natural number n, it is NP-complete to decide whether or not there is a MinA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ of cardinality $\leq n$.*

Another property of interest when trying to understand a c-property $\mathcal{P}$ is whether a given axiom t is *relevant* for $\mathcal{P}$; that is, whether there is a MinA that contains t. This knowledge is helpful, for instance, when trying to compute the set of all MinAs. In [KPHS07], the authors propose the use of Reiter's Hitting Set Tree algorithm [Rei87] as an improved black-box algorithm for producing the set of all MinAs. This idea has since then been used and further optimised for specific decision problems [SHCH07, BS08, SQJH08]. Detecting axioms that are relevant would allow us to further improve this approach using the set enumeration procedure proposed by Rymon [Rym92]. The following theorem shows that deciding relevance of axioms is also an NP-hard problem.

Theorem 6.2. *Let $\mathcal{T}$ be a $\mathcal{HL}$ TBox, $t \in \mathcal{T}$, and A, B two concept names appearing in $\mathcal{T}$. Deciding whether there exists a MinA $\mathcal{S}$ for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ such that $t \in \mathcal{S}$ is NP-complete in the size of $\mathcal{T}$.*

[22] This strategy corresponds to the naïve algorithm presented in [BPS07a, Sun09]

Proof. The problem is clearly in NP as we need only polynomial time to test whether a set of axioms $\mathcal{S}$ is a MinA, and whether $t \in \mathcal{S}$. The complexity hardness can be shown by a reduction of the following NP-complete problem [FGN90, EG95a]: given two sets of propositional variables H, M, a set T of definite Horn clauses over $H \cup M$ (i.e., formulae of the form $v_1 \wedge \ldots \wedge v_n \rightarrow w$ with $w, v_i \in H \cup M$ for all $1 \leq i \leq n$), and a variable $h \in H$, decide whether there is a *minimal* $H' \subseteq H$ such that $h \in H'$ and $H' \cup T \models M$.

Given an instance of this problem, we define a concept name P_i for every $h_i \in H$ and Q_i for every $m_i \in M$; additionally, we use two new concept names A, B. Our TBox has an axiom of the form $A \sqsubseteq P_i$ for every $h_i \in H$, an axiom $R_1 \sqcap \ldots \sqcap R_n \sqsubseteq R$ for every $v_1 \wedge \ldots \wedge v_n \rightarrow w \in T$, and additionally the axiom $\bigsqcap_{m_i \in M} Q_i \sqsubseteq B$. It is easy to see that, given a variable $h_0 \in H$, there is a MinA for $A \sqsubseteq B$ containing $A \sqsubseteq P_0$ iff there is a minimal $H' \in H$ such that $h_0 \in H'$ and $H' \cup T \models M$. □

As it was already said, finding the lexicographical last MinA for subsumption of $\mathcal{HL}$ concept names requires only polynomial time. If, on the contrary, we are interested in finding the lexicographical *first* MinA, then we encounter another hard problem.

Theorem 6.3. *Given an $\mathcal{HL}$ TBox $\mathcal{T}$, concept names A, B occurring in $\mathcal{T}$ and a MinA $\mathcal{S}$, it is coNP-complete to tell whether $\mathcal{S}$ is the lexicographical first MinA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$.*

Proof. The problem is in coNP since if S is not the lexicographical first, then we can prove this by presenting a MinA that appears before S within this ordering. To prove hardness, we will make a reduction from the *first lexicographical minimal vertex cover* problem. Given a graph $G = (V, E)$, a set $C \subseteq V$ is called a *vertex cover* if for every edge $(u, v) \in E$ either $u \in C$ or $v \in C$. For a graph G and a minimal vertex cover D, it is coNP-complete to decide whether D is the first lexicographical minimal vertex cover [JYP88]. Alternatively, we can see this problem as deciding the first lexicographical hitting set from a collection of sets of cardinality at most two. Suppose that $V = \{v_1, \ldots, v_n\}$ and that $E = \{e_1, \ldots, e_k\}$ where for every $i, 1 \leq i \leq k$, e_i is of the form $e_i = \{v, w\}$. We use a concept name P_i for every element $v_i \in V$, a concept name Q_j for every edge in $e_j \in E$ and the additional concept names A, B, and define the TBox

$$\begin{aligned} \mathcal{T} \;:=\; & \{A \sqsubseteq P_i \mid 1 \leq i \leq n\} \cup \\ & \{P_i \sqsubseteq \bigsqcap_{v_i \in e_j} Q_j \mid 1 \leq i \leq n\} \cup \{Q_1 \sqcap \ldots \sqcap Q_k \sqsubseteq B\}. \end{aligned}$$

Hence, there are $2n + 1$ axioms, which we order in the following way: for $1 \leq m \leq n$, the m-th axiom is $A \sqsubseteq P_m$ and the $n+m$-th axiom is $P_m \sqsubseteq \bigsqcap_{v_i \in e_j} Q_j$, with $\bigsqcap_{j=1}^{k} Q_k \sqsubseteq B$ as the last axiom. If D is a minimal vertex cover, then the set

$$S_D = \{A \sqsubseteq P_i, P_i \sqsubseteq \bigsqcap_{v_i \in e_j} Q_j \mid v_i \in D\} \cup \{\bigsqcap_{j=1}^{k} Q_j \sqsubseteq B\}$$

is a MinA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$. Additionally, if S is a MinA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$, then S satisfies the following two properties: (i) $\bigsqcap_{j=1}^{k} Q_j \sqsubseteq B \in S$, and (ii) $A \sqsubseteq P_i \in S$ iff $P_i \sqsubseteq \bigsqcap_{v_i \in e_j} Q_j \in S$ for all $1 \leq i \leq n$. Thus, for every MinA S we can construct the set $D = \{v_i \mid A \sqsubseteq P_i \in S\}$, which is such that $S = S_D$. Furthermore, the way the ordering was defined ensures that a D is lexicographically before D' if and only if S_D is lexicographically before $S_{D'}$. This means that D is the lexicographical first minimal vertex cover iff S_D is the lexicographical first MinA. □

Since the decision problems we have presented in this section depend, in a greater or smaller degree, on the set of all MinAs, it could be argued that their hardness is a consequence of the fact that an axiomatic input can have exponentially many MinAs (see [Sun09, BPS07a] for an example). We could instead try to analyse the complexity of enumerating the set of all MinAs [JYP88]. An algorithm that enumerates all MinAs using time polynomial in the size of both the input *and the output*, that is, in the size of the TBox and the number of MinAs, will be called *output polynomial*. If we had an output polynomial algorithm, then its execution time would be polynomial whenever the axiomatic input had only polynomially many MinAs.

We are interested in the enumeration complexity of computing the set of all MinAs for an $\mathcal{HL}$-TBox w.r.t. a given subsumption relation. Unfortunately, to the best of our efforts we were unable to find a tight bound on the complexity of this problem. Hence, we settle here for weaker results, in which we allow additional expressivity in the ontology. We will show that if we either allow a set of irrefutable axioms, or if we extend the syntax of axioms to allow disjunction in the left-hand size, then an output polynomial algorithm computing all MinAs is impossible.

Before proving this, we will present an auxiliary result showing that it is not possible to enumerate all the minimal valuations satisfying a monotone Boolean formula with an output polynomial algorithm. A proof of this fact can be found in the technical report [EG91]; since this result is not included in the corresponding journal paper [EG95b], we provide our own distinct proof for the sake of completeness.

Theorem 6.4. *There is no output polynomial algorithm for computing all minimal satisfying valuations of monotone Boolean formulae, unless P=NP.*

To prove this theorem, it is enough to show (see [KSS00]) that the following decision problem is NP-hard:

Lemma 6.5. *Given a monotone Boolean formula ϕ and a set $\mathcal{M}$ of minimal valuations satisfying ϕ, deciding whether there exists a minimal valuation $\mathcal{V} \notin \mathcal{M}$ satisfying ϕ is NP-hard in the size of ϕ and $\mathcal{M}$.*

Proof. The proof is by reduction of the NP-hard *hypergraph 2-coloring problem* [GJ79]: given a collection $H = \{E_1, \ldots, E_m\}$ of subsets of a set of vertices V, each of them of size 3, is there a set C such that $C \cap E_i \neq \emptyset$ and $(V \setminus C) \cap E_i \neq \emptyset$ for $i = 1, \ldots, m$.[23]

[23] In other words, both C and its complement must be hitting sets for $E_1, \ldots, E_m$.

Let $V = \{v_1, \ldots, v_n\}$ and $E_i = \{v_{i1}, v_{i2}, v_{i3}\}$ for all $i = 1, \ldots, m$. We represent every $v_i \in V$ by a propositional variable p_i, and construct the monotone Boolean formula $\phi := \psi \vee \bigvee_{i=1}^{m} \psi_i$, where

$$\psi = \bigwedge_{i=1}^{m} p_{i1} \vee p_{i2} \vee p_{i3} \quad \text{and} \quad \psi_i = p_{i1} \wedge p_{i2} \wedge p_{i3}$$

and the set

$$\mathcal{M} := \{\mathcal{V}_i := \{p_{i1}, p_{i2}, p_{i3}\} \mid 1 \leq i \leq m \text{ and no strict subset of } \mathcal{V}_i \text{ satisfies } \psi\}.$$

It is easy to see that the formula ϕ as well as the set $\mathcal{M}$ can be constructed in time polynomial in the size of V and H. Moreover, every valuation $\mathcal{V}_i \in \mathcal{M}$ satisfies the formula ψ_i, and hence also ϕ. It is minimal since no strict subset of $\mathcal{V}_i$ satisfies (i) any of the ψ_j (which require valuations of size at least 3 to be satisfied) nor (ii) ψ since otherwise the condition in the definition of $\mathcal{M}$ would be violated. This shows that ϕ and $\mathcal{M}$ indeed form an instance of the problem considered in the lemma.

To complete the proof of NP-hardness of this problem, it remains to be shown that there is a minimal valuation $\mathcal{V} \notin \mathcal{M}$ satisfying ϕ iff there is a set $C \subseteq V$ such that $C \cap E_i \neq \emptyset$ and $(V \setminus C) \cap E_i \neq \emptyset$ for all $1 \leq i \leq m$.

We show first the *if direction*. Let C be such a set, which we assume without loss of generality to be minimal with respect to set inclusion. We define the valuation $\mathcal{V}_C := \{p_i \mid v_i \in C\}$ and claim that it is the minimal valuation we are looking for. For every $1 \leq i \leq m$, $C \cap E_i \neq \emptyset$ implies that there is a $1 \leq j \leq 3$ such that $v_{ij} \in C$, which means that $p_{ij} \in \mathcal{V}_C$. This shows that $\mathcal{V}_C$ satisfies ψ and thus also ϕ. In addition, since $(V \setminus C) \cap E_i \neq \emptyset$, there is a $1 \leq k \leq 3$ such that $v_{ik} \notin C$. Thus, $\mathcal{V}_C$ is different from all the valuations $\mathcal{V}_i \in \mathcal{M}$, and it does not satisfy any of the formulae ψ_i.

To show that $\mathcal{V}_C$ is minimal, assume that $\mathcal{V}' \subset \mathcal{V}_C$. Since C is minimal, the set $C' := \{v_i \mid p_i \in \mathcal{V}'\} \subset C$ is such that there is a $1 \leq i \leq m$ with $C' \cap E_i = \emptyset$. This implies that $\mathcal{V}'$ does not satisfy $p_{i1} \vee p_{i2} \vee p_{i3}$, and hence it does not satisfy ψ. As a subset of $\mathcal{V}_C$, it also does not satisfy any of the formulae ψ_i, and thus it does not satisfy ϕ. This shows that $\mathcal{V}_C$ is a minimal valuation satisfying ϕ that does not belong to $\mathcal{M}$.

For the *only-if direction*, assume that there is a minimal valuation $\mathcal{V} \notin \mathcal{M}$ satisfying ϕ. This valuation cannot satisfy any of the formulae ψ_i. Indeed, (i) for $\mathcal{V}_i \in \mathcal{M}$ this would imply that $\mathcal{V}$ is a superset of one of the valuations in $\mathcal{M}$, which contradicts either the minimality of $\mathcal{V}$ or the fact that it does not belong to $\mathcal{M}$; (ii) for $\mathcal{V}_i \notin \mathcal{M}$ there would be a smaller valuation satisfying ψ, which contradicts the minimality of $\mathcal{V}$.

Since $\mathcal{V}$ is a model of ϕ, it must thus satisfy ψ. Define the set $C_{\mathcal{V}} := \{v_i \mid p_i \in \mathcal{V}\}$. Since $\mathcal{V}$ satisfies ψ, for every $1 \leq i \leq m$ there is a $1 \leq j \leq 3$ such that $p_{ij} \in \mathcal{V}$, and thus $v_{ij} \in C_{\mathcal{V}} \cap E_i$. On the other hand, since $\mathcal{V}$ does not satisfy any of the formulae ψ_i, for every $1 \leq i \leq m$ there must also be a $1 \leq k \leq 3$ such that $p_{ik} \notin \mathcal{V}$, which means that $E_i \not\subseteq C_{\mathcal{V}}$ and hence $(V \setminus C) \cap E_i \neq \emptyset$. □

Theorem 6.4 follows from this lemma since an output polynomial algorithm whose runtime is bounded by the polynomial $P(|\phi|, |\mathcal{M}|)$ (where ϕ is the input and $\mathcal{M}$ the

output) could be used to decide the problem introduced in the lemma in polynomial time as follows: given ϕ and $\mathcal{M}$, run the algorithm for time at most $P(|\phi|, |\mathcal{M}|)$ and check whether the generated valuations are exactly those in $\mathcal{M}$.

Theorem 6.4 shows that an algorithm for computing all MinAs based on computing the pinpointing formula and then producing its minimal satisfying valuations cannot be output polynomial. We would like to show that there is no algorithm of any kind for computing all MinAs that is output polynomial. Unfortunately, our efforts towards this goal have been unfruitful. In this respect, we had to settle with a weaker result dealing with the enumeration of all MinAs in the presence of an *irrefutable* TBox. Assume that the TBox is formed by the disjoint union of a *static* TBox $\mathcal{T}_s$ whose axioms are irrefutable, and a *refutable* TBox $\mathcal{T}_r$. We will denote this union as $\mathcal{T} = (\mathcal{T}_s \uplus \mathcal{T}_r)$. In this context, a MinA $\mathcal{S}$ for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ is a minimal subset of $\mathcal{T}_r$ such that $A \sqsubseteq_{\mathcal{T}_s \cup \mathcal{S}} B$. In Chapter 3 we showed that this defines a c-property, and hence we can use the notions of MinA in it.

Theorem 6.6. *There is no output polynomial algorithm that computes, for a given $\mathcal{HL}$ TBox $\mathcal{T} = (\mathcal{T}_s \uplus \mathcal{T}_r)$ and concept names A, B occurring in $\mathcal{T}$, all MinAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$, unless P=NP.*

Proof. We show that the problem of computing the minimal valuations of monotone Boolean formulae can be reduced in polynomial time to the problem of computing the MinAs of an $\mathcal{HL}$ TBox. Given a monotone Boolean formula ϕ, we introduce one concept name B_ψ for every subformula of ψ of ϕ, and one additional concept name A. We define TBoxes $\mathcal{T}_\psi$ for the subformulae ψ of ϕ by induction in a straightforward manner: if $\psi = p$ is a propositional variable, then $\mathcal{T}_\psi := \{A \sqsubseteq B_p\}$; if $\psi = \psi_1 \wedge \psi_2$, then $\mathcal{T}_\psi := \{B_{\psi_1} \sqcap B_{\psi_2} \sqsubseteq B_\psi\}$; if $\psi = \psi_1 \vee \psi_2$, then $\mathcal{T}_\psi := \{B_{\psi_1} \sqsubseteq B_\psi, B_{\psi_2} \sqsubseteq B_\psi\}$.

Obviously, the size of $\mathcal{T}_\phi$ is linear in the size of ϕ. In $\mathcal{T}_\phi$, we declare the GCIs $A \sqsubseteq B_p$ with p a propositional variable to be refutable, and the other GCIs to be irrefutable. With this division of $\mathcal{T}_\phi$ into a static and a refutable part, it is easy to see that there is a one-to-one correspondence between the minimal satisfying valuations of ϕ and the MinAs for $\mathcal{T}_\phi$ w.r.t. $A \sqsubseteq B_\phi$. In particular, given a MinA $\mathcal{S}$, the corresponding valuation $\mathcal{V}_\mathcal{S}$ consists of all p such that $A \sqsubseteq B_p \in \mathcal{S}$. Thus, if we could compute all MinAs with an output polynomial algorithm, we could do the same for all minimal satisfying valuations. □

This theorem shows that, in general, exponential time is necessary for computing all the MinAs of a given axiomatic input, even if there are only polynomially many of them, when some of the axioms are allowed to be irrefutable. The reason why irrefutable axioms are necessary is to be able to adequately model the disjunctions from which we are reducing the problem. It seems reasonable, thus, that if we allow the language to include the disjunction constructor ($\sqcup$), then there will be no need for a static TBox. We will now show that it suffices to allow this constructor only on the left-hand side of the axioms. More formally, we define the set of $\mathcal{HLU}$ concept terms as those that can be obtained from the set $\mathsf{N_C}$ of concept names using the constructors $\sqcap$ and $\sqcup$. A *disjunctive TBox* is a set of axioms of the form $C \sqsubseteq D$ where C is an

$\mathcal{HLU}$ concept term and D is an $\mathcal{HL}$ concept term. The semantics of this logic are defined in the obvious way.

Theorem 6.7. *Let $\mathcal{T}$ be a disjunctive TBox and A, B two concept names appearing in $\mathcal{T}$. There is no output polynomial algorithm for computing all MinAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$, unless $P = NP$.*

Proof. The proof is very similar to that of Theorem 6.4 through Lemma 6.5. We will reduce the hypergraph 2-coloring to the problem of deciding, given a set of MinAs $\mathcal{M}$, whether there is another MinA for our property that is not an element of $\mathcal{M}$.

Let $V = \{v_1, \ldots, v_n\}$ and $E_i = \{v_{i1}, v_{i2}, v_{i3}\}$ for $i, 1 \leq i \leq m$. We will simulate each $v_i \in V$ by a concept name P_i. If we define the axiom t_B as

$$t_B := \prod_{i=1}^{m}(p_{i1} \sqcup p_{i2} \sqcup p_{i3}) \sqcup \bigsqcup_{i=1}^{m}(p_{i1} \sqcap p_{i2} \sqcap p_{i3}) \sqsubseteq B,$$

then we construct the disjunctive TBox $\mathcal{T} = \{A \sqsubseteq P_i \mid 1 \leq i \leq n\} \cup \{t_B\}$, and the set of MinAs

$$\mathcal{M} := \{\mathcal{V}_i := \{A \sqsubseteq P_{ij} \mid 1 \leq j \leq 3\} \cup \{t_B\} \mid 1 \leq i \leq m \text{ and } \mathcal{V}_i \text{ is a MinA}\}.$$

Since the concept name B appears only in the right-hand side of the axiom t_B, any MinA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ must contain this axiom. Thus, using an argument analogous to the one of Lemma 6.5, we have that there is a MinA $\mathcal{S} \notin \mathcal{M}$ for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ if and only if there is a set $C \sqsubseteq V$ such that $C \cap E_i \neq \emptyset$ and $(V \setminus C) \cap E_i \neq \emptyset$ for all $i, 1 \leq i \leq m$. From this result, our claim follows, using the same argument as in the proof of Theorem 6.4. □

Alternatively one may be interested in knowing how many MinAs there are, rather than actually obtaining each of them. For these kind of problems, where the interest is in counting the number of solutions, we have to analyse a different kind of complexity. In the theory of counting complexity, given a decision problem, one is not only interested in whether there is a solution or not, but rather in how many solutions exist. Clearly, the resources necessary for counting the number of solutions exceed those needed for merely deciding the existence of one since any number of solutions greater to zero implies an affirmative answer to the decision problem. In the first papers introducing this complexity measure, Valiant showed that there exist problems decidable in polynomial time for which counting the number of solutions is as hard as for NP-complete problems [Val79a, Val79b]. Informally, the counting complexity class #P contains all those problems for which a solution to its related decision problem can be verified in polynomial time. Thus, the counting problem of every decision problem in NP belongs to #P.

Theorem 6.8. *Given a $\mathcal{HL}$ TBox $\mathcal{T}$ and two concept names A, B occurring in $\mathcal{T}$, the problem of counting the number of MinAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ is #P-complete.*

Proof. The problem is in #P since its underlying decision problem, whether there exist a MinA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ is in NP.[24] We show #P-hardness by a reduction

[24] Actually, as it has already been said, it is in P.

of the #P-hard *minimal vertex cover counting problem* [Val79b]: given a set V and $E \subseteq V \times V$, count the number of minimal vertex covers. In other words, counting the number of minimal hitting sets of a collection of sets of cardinality at most two. We use the same reduction presented in the proof of Theorem 6.3, and show that it is *parsimonious*; i.e. that is preserves the number of solutions. As shown in said proof, a set $C \subseteq V$ is a minimal set having at least one element of each $e \in E$ iff $\mathcal{S}_C := \{A \sqsubseteq P_i, P_i \sqsubseteq \bigsqcap_{v_i \in e_j} Q_j \mid v_i \in C\} \cup \{\bigsqcap_{j=1}^{k} Q_j \sqsubseteq B\}$ is a MinA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$. We have thus a one-to-one correspondence between the number of vertex covers and the number of MinAs. Hence, counting the number of MinAs is a #P-hard problem. □

Another interesting question regarding counting is, given an axiom t, compute the number of MinAs that have t as an element. Solving this problem is relevant, for example, when correcting an unwanted consequence: those axioms that appear more often as causes of the error are the most likely to be faulty, and their removal will also eliminate the most MinAs possible. This idea has been proposed as an heuristic for correcting an error while minimizing the changes in the set of axioms [Sch05, SHCH07]. Unfortunately, this counting problem is also #P-hard.

Theorem 6.9. *Given an $\mathcal{HL}$ TBox $\mathcal{T}$, an axiom $t \in \mathcal{T}$, and two concept names A, B occurring in $\mathcal{T}$, the problem of counting the number of MinAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ containing t is #P-complete.*

Proof. This problem is in #P as its underlying decision problem is in NP. We show #P-hardness by giving a parsimonious reduction of the problem from Theorem 6.8. Given an $\mathcal{HL}$ TBox $\mathcal{T}$ and two concept names A, B appearing in $\mathcal{T}$, we define the new $\mathcal{HL}$ TBox $\mathcal{T}' := \mathcal{T} \cup \mathcal{S}_0$, where $\mathcal{S}_0 = \{A \sqsubseteq C, B \sqcap C \sqsubseteq D\}$ and C and D are concept names not occurring in $\mathcal{T}$. Clearly, a set $\mathcal{S} \subseteq \mathcal{T}$ is a MinA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ iff $\mathcal{S} \cup \mathcal{S}_0$ is a MinA for $\mathcal{T}'$ w.r.t. $A \sqsubseteq D$. Furthermore, every MinA for $\mathcal{T}'$ w.r.t. $A \sqsubseteq D$ must contain the axioms in $\mathcal{S}_0$. Thus, there are exactly as many MinAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ as there are MinAs for $\mathcal{T}'$ w.r.t. $A \sqsubseteq D$ containing the axiom $A \sqsubseteq C$, which entails the hardness result. □

With this result we finish our study of complexity of problems related to finding MinAs. In the following subsection we will show that the same complexity bounds hold for the dual problems related to MaNAs.

6.1.2 MaNA Complexity

Finding minimal hitting sets has been useful, not only when trying to produce the set of all MaNAs from known MinAs and *vice versa*, but also to prove the hardness of MinA related problems in the previous subsection. Given the dual nature of MinAs and MaNAs, it is hardly surprising that the dual problem of hitting sets – that of independent sets – will be equally helpful for showing the hardness of MaNA related problems.

Algorithm 1 Compute one MaNA for $\mathcal{T} = \{t_1, \ldots, t_n\}$ w.r.t. $A \sqsubseteq B$.

```
1: if A ⋢_T B then
2:    return no MaNA
3: S := ∅
4: for 1 ≤ i ≤ n do
5:    if A ⋢_{S∪{t_i}} B then
6:       S := S ∪ {t_i}
7: return S
```

Given a collection $\mathcal{M}$ of sets using elements from $\mathcal{V}$, a set $\mathcal{S} \subseteq \mathcal{V}$ is an *independent set* iff for every $M \in \mathcal{M}$ it holds that $M \not\subseteq \mathcal{S}$. Notice that $\mathcal{S}$ is a (maximal) independent set if and only if $\mathcal{V} \setminus \mathcal{S}$ is a (minimal) hitting set. Thus, all complexity results known for (minimal) hitting sets apply also, in their dual presentation, to (maximal) independent sets, and likewise for the opposite direction. This is, nonetheless, not sufficient for claiming that all the results from Section 6.1.1 hold also for MaNAs, since the c-properties considered change with this polynomial reduction.

Although not all of the algorithms known for computing a single MinA can be dualised, we can still compute one MaNA – in fact, the lexicographical first MaNA – with only a polynomial overhead, by dualising the naive algorithm presented in [Sun09, BPS07a] in such a way that adds axioms to the knowledge base, as long as the property does not follow from the enlarged set. This dual version, for the case of subsumption w.r.t. $\mathcal{HL}$-TBoxes, is shown in Algorithm 1. This algorithm requires polynomially many subsumption tests. Furthermore, it is easy to see that its output corresponds to the first lexicographical MaNA.

If the search for a MaNA aims to avoiding an unwanted consequence, then we are interested in finding the largest possible MaNA, that is, one with the greatest cardinality, such that the changes to the knowledge base remain minimal. Deciding whether there is a MaNA of size greater than or equal to a given n is an NP-complete problem, though.

Theorem 6.10. *Given an $\mathcal{HL}$ TBox $\mathcal{T}$, concept names A, B appearing in $\mathcal{T}$ and a natural number n, it is NP-complete to decide the existence of a MaNA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ of cardinality $\geq n$.*

Proof. The problem is obviously in NP. For the hardness, we reduce the NP-hard problem of *maximal independent sets*: given a collection of sets $\mathcal{M} = \{S_1, \ldots, S_k\}$ and a natural number n, decide whether there is an independent set for $\mathcal{M}$ of cardinality $\geq n$. For the reduction, we use a concept name P for every element $p \in \bigcup_{i=1}^{k} S_i$ and additional concept names A, B. We consider that each set S_i is of the form $S_i = \{s_{i1}, \ldots, s_{i\ell_i}\}$ and construct the TBox:

$$\begin{aligned} \mathcal{T} \;:=\; & \{A \sqsubseteq P \mid p \in \textstyle\bigcup_{i=1}^{k} S_i\} \cup \\ & \{\prod_{j=1}^{\ell_i} P_{ij} \sqsubseteq B \mid 1 \leq i \leq k\} \end{aligned}$$

We will show that there is an independent set for $\mathcal{M}$ of size $\geq n$ iff there is a MaNA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ of size $\geq n + k$.

Assume first that there is such an independent set M. The sub-TBox

$$\mathcal{T}' = \{A \sqsubseteq P \mid p \in M\} \cup \mathcal{S}$$

where

$$\mathcal{S} \quad := \quad \{\prod_{j=1}^{\ell_i} P_{ij} \sqsubseteq B \mid 1 \leq i \leq k\} \tag{6.1}$$

has $|M| + k$ axioms and is such that $A \not\sqsubseteq_{\mathcal{T}'} B$.

Conversely, take a MaNA $\mathcal{T}'$. Suppose that there is a $i, 1 \leq i \leq k$ such that $\prod_{j=1}^{\ell_i} P_{ij} \sqsubseteq B \notin \mathcal{T}'$. Since $\mathcal{T}'$ is a MaNA, it holds that $\{A \sqsubseteq P_{ij} \mid 1 \leq j \leq \ell_i\} \subseteq \mathcal{T}'$. Take now any element from S_i; say p_{i1}. Then, the new sub-TBox

$$\mathcal{T}'_i = (\mathcal{T}' \setminus \{A \sqsubseteq P_{i1}\}) \cup \{\prod_{j=1}^{\ell_i} P_{ij} \sqsubseteq B\}$$

is such that (i) $|\mathcal{T}'_i| = |\mathcal{T}'|$, and (ii) $A \not\sqsubseteq_{\mathcal{T}'_i} B$. The same process can be applied again to this set $\mathcal{T}'_i$, until we have constructed a set of axioms $\mathcal{T}''$ such that $A \not\sqsubseteq_{\mathcal{T}''} B$ and $\mathcal{S} \subseteq \mathcal{T}''$, where $\mathcal{S}$ is the one of Equation (6.1). The set $M = \{p \mid A \sqsubseteq P \in \mathcal{T}''\}$ is an independent set for $\mathcal{M}$, and $|\mathcal{T}''| = |M| + k$. □

Just as we were interested in the relevance of an axiom when dealing with MinAs, one might want to know whether a given axiom *necessarily* appears in every MaNA, or there is at least one that does not contain it. We show that this problem is equivalent to that of Theorem 6.2.

Theorem 6.11. *Let $\mathcal{T}$ be a $\mathcal{HL}$ TBox, $t \in \mathcal{T}$ and A, B concept names in $\mathcal{T}$. Deciding the existence of a MaNA $\mathcal{S}$ for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ such that $t \notin \mathcal{S}$ is NP-complete on the size of $\mathcal{T}$.*

Proof. Let $\mathcal{S}$ be a MaNA such that $t \notin \mathcal{S}$. Then, for $\mathcal{S} \cup \{t\}$ it holds that $A \sqsubseteq_{\mathcal{S} \cup \{t\}} B$. Thus, there is a MinA $\mathcal{S}'$ for $A \sqsubseteq B$ w.r.t. $\mathcal{T}$ such that $\mathcal{S}' \subseteq \mathcal{S} \cup \{t\}$. But then, it holds that $t \in \mathcal{S}'$ since otherwise $\mathcal{S}' \subseteq \mathcal{S}$, which would contradict the fact that $\mathcal{S}'$ is a MinA. Conversely, if $\mathcal{S}$ is a MinA such that $t \in \mathcal{S}$, then the subsumption relation does not hold for $\mathcal{S} \setminus \{t\}$. Hence, there is a MaNA $\mathcal{S}'$ containing $\mathcal{S} \setminus \{t\}$. If $t \in \mathcal{S}'$, then $\mathcal{S} \subseteq \mathcal{S}'$, contradicting the definition of MaNA. Hence, there is a MinA containing t if and only if there is a MaNA that does not contain t. □

To finish with the decision complexity results, we show coNP-hardness for the problem of finding the lexicographical *last* MaNA. This follows easily from the hardness of finding the lexicographical last maximal independent set.

Theorem 6.12. *Given an $\mathcal{HL}$ TBox $\mathcal{T}$, concept names A, B appearing in $\mathcal{T}$ and a MaNA $\mathcal{S}$, it is coNP-complete to tell whether $\mathcal{S}$ is the lexicographical last MaNA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$.*

Proof. The problem is in coNP since we can verify a counterexample in polynomial time. For the hardness, we use the result from [JYP88] by which finding the lexicographical last maximal independent set is coNP-hard. We use the same reduction from the proof of Theorem 6.10 and order the axioms as follows: first all the axioms of the form $A \sqsubseteq P$, and then all the other axioms. It is easy to see that M is the last lexicographical maximal independent set for $\mathcal{M}$ if and only if $\mathcal{T}' = \{A \sqsubseteq P_j \mid p_j \in M\} \cup \mathcal{S}$, with $\mathcal{S}$ as in Equation (6.1), is the last lexicographical MaNA for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$. □

We focus now on the complexity of enumerating all MaNAs. For a fixed natural number n, consider the $\mathcal{HL}$ TBox

$$\mathcal{T}_n = \{A \sqsubseteq P_i, P_i \sqsubseteq B \mid 1 \leq i \leq n\}.$$

$\mathcal{T}_n$ has $2n$ axioms, but for every set $N \subseteq \{1, \ldots, n\}$, the sub-TBox

$$\{A \sqsubseteq P_i \mid i \in N\} \cup \{P_j \sqsubseteq B \mid j \notin N\}$$

is a MaNA for $\mathcal{T}_n$ w.r.t. $A \sqsubseteq B$. Since each different N defines a different MaNA, this axiomatic input has 2^n MaNAs. This example shows that a given axiomatic input may have exponentially many MaNAs, measured on the number of axioms. We will show that they cannot be enumerated using an output polynomial algorithm, in the presence of an irrefutable TBox. As it was the case for MinAs, we will show first an auxiliary result regarding the computation of all maximal valuations falsifying a monotone Boolean formula.

Lemma 6.13. *Given a monotone Boolean formula ϕ and a set $\mathcal{M}$ of maximal valuations falsifying ϕ, deciding whether there exists a maximal valuation $\mathcal{V} \notin \mathcal{M}$ falsifying ϕ is NP-hard in the size of ϕ and $\mathcal{M}$.*

Proof. For the proof, we once again use the NP-hard hypergraph 2-coloring problem. Our reduction in this case will be very similar to the one used in Lemma 6.5, taking advantage of the duality of the problems. Let $V = \{v_1, \ldots, v_n\}$ and $E_i = \{v_{i1}, v_{i2}, v_{i3}\}$ for all $i = 1, \ldots, m$. We represent every $v_i \in V$ by a propositional variable p_i, call $\mathcal{P}$ the set of all propositional variables representing a $v \in V$. and construct the monotone Boolean formula $\phi := \psi \wedge \bigwedge_{i=1}^{m} \psi_i$, where

$$\psi = \bigvee_{i=1}^{m} p_{i1} \wedge p_{i2} \wedge p_{i3} \quad \text{and} \quad \psi_i = p_{i1} \vee p_{i2} \vee p_{i3}$$

and the set

$$\mathcal{M} := \{\mathcal{V}_i := \mathcal{P} \setminus \{p_{i1}, p_{i2}, p_{i3}\} \mid 1 \leq i \leq m \text{ and no strict superset of } \mathcal{V}_i \text{ falsifies } \psi\}.$$

It is easy to see that the formula ϕ as well as the set $\mathcal{M}$ can be constructed in time polynomial in the size of V and H. Moreover, every valuation $\mathcal{V}_i \in \mathcal{M}$ falsifies the formula ψ_i, and hence also ϕ. It is maximal since no strict superset of $\mathcal{V}_i$ falsifies (i) any of the ψ_j (which require valuations of size at most $n - 3$ to be falsified) nor

(ii) ψ since otherwise the condition in the definition of $\mathcal{M}$ would be violated. This shows that ϕ and $\mathcal{M}$ indeed form an instance of the problem considered in the lemma.

To complete the proof of NP-hardness of this problem, it remains to be shown that there is a maximal valuation $\mathcal{V} \notin \mathcal{M}$ falsifying ϕ iff there is a set $C \subseteq V$ such that $C \cap E_i \neq \emptyset$ and $(V \setminus C) \cap E_i \neq \emptyset$ for all $1 \leq i \leq m$.

We show first the *if direction*. Let C be such a set, which we assume without loss of generality to be minimal with respect to set inclusion. We define the valuation $\mathcal{V}_C := \mathcal{P} \setminus \{p_i \mid v_i \in C\}$ and claim that it is the maximal valuation we are looking for. For every $1 \leq i \leq m$, $C \cap E_i \neq \emptyset$ implies that there is a $1 \leq j \leq 3$ such that $v_{ij} \in C$, which means that $p_{ij} \notin \mathcal{V}_C$. This shows that $\mathcal{V}_C$ falsifies ψ and thus also ϕ. In addition, since $(V \setminus C) \cap E_i \neq \emptyset$, there is a $1 \leq k \leq 3$ such that $v_{ik} \in C$. Thus, $\mathcal{V}_C$ is different from all the valuations $\mathcal{V}_i \in \mathcal{M}$, and it satisfies all of the formulae ψ_i.

To show that $\mathcal{V}_C$ is maximal, assume that $\mathcal{V}' \supset \mathcal{V}_C$. Since C is minimal, the set $C' := \{v_i \mid p_i \notin \mathcal{V}'\} \subset C$ is such that there is a $1 \leq i \leq m$ with $C' \cap E_i = \emptyset$. This implies that $\mathcal{V}'$ satisfies $p_{i1} \wedge p_{i2} \wedge p_{i3}$, and hence it also satisfies ψ. As a superset of $\mathcal{V}_C$, it also satisfies all of the formulae ψ_i, and thus it must satisfy ϕ. This shows that $\mathcal{V}_C$ is a maximal valuation falsifying ϕ that does not belong to $\mathcal{M}$.

For the *only-if direction*, assume that there is a maximal valuation $\mathcal{V} \notin \mathcal{M}$ falsifying ϕ. This valuation cannot falsify any of the formulae ψ_i. Indeed, (i) for $\mathcal{V}_i \in \mathcal{M}$ this would imply that $\mathcal{V}$ is a subset of one of the valuations in $\mathcal{M}$, which contradicts either the maximality of $\mathcal{V}$ or the fact that it does not belong to $\mathcal{M}$; (ii) for $\mathcal{V}_i \notin \mathcal{M}$ there would be a larger valuation falsifying ψ, which contradicts the maximality of $\mathcal{V}$.

Since $\mathcal{V}$ is not a model of ϕ, it must thus falsify ψ. Define for this valuation the set $C_\mathcal{V} := \{v_i \mid p_i \notin \mathcal{V}\}$. Since $\mathcal{V}$ falsifies ψ, for every $1 \leq i \leq m$ there is a $1 \leq j \leq 3$ such that $p_{ij} \notin \mathcal{V}$, and thus $v_{ij} \in C_\mathcal{V} \cap E_i$. On the other hand, since $\mathcal{V}$ does not falsify any of the formulae ψ_i, for every $1 \leq i \leq m$ there must also be a $1 \leq l \leq 3$ such that $p_{i,k} \in \mathcal{V}$, which means that $E_i \not\subseteq C_\mathcal{V}$ and hence $(V \setminus C) \cap E_i \neq \emptyset$. □

From this lemma, we get the following theorem, whose proof is identical to the one for Theorem 6.4 presented in the previous subsection.

Theorem 6.14. *There is no output polynomial algorithm for computing all maximal falsifying valuations of monotone Boolean formulae, unless P=NP.*

In the proof of Theorem 6.6, we presented a one to one correspondence between MinAs using an irrefutable TBox, and minimal valuations satisfying a Boolean formula. It is easy to see that the same reduction yields also a bijection between the set of MaNAs for the same property and the maximal valuations falsifying the formula. We thus obtain the next result.

Theorem 6.15. *There is no output polynomial algorithm that computes, for a given $\mathcal{HL}$ TBox $\mathcal{T} = (\mathcal{T}_s \uplus \mathcal{T}_r)$ and concept names A, B occurring in $\mathcal{T}$, all MaNAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$, unless P=NP.*

Recall now that it is possible to simulate a monotone Boolean formula using a disjunctive TBox. Thus, the dual result for Theorem 6.7 holds too.

Theorem 6.16. *Let $\mathcal{T}$ be a disjunctive TBox and A, B two concept names appearing in $\mathcal{T}$. There is no output polynomial algorithm for computing all MaNAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$, unless $P = NP$.*

Considering now the problem of counting the number of solutions, we get the two results that counting the number of MaNAs and the number of all MaNAs not containing a given axiom t are #P-complete problems.

Theorem 6.17. *Given a $\mathcal{HL}$ TBox $\mathcal{T}$, two concept names A, B appearing in $\mathcal{T}$ and an axiom $t \in \mathcal{T}$, the following two problems are #P-complete:*

1. *counting the number of MaNAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$;*
2. *counting the number of MaNAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ not containing t.*

Proof. For the first point, consider $\mathcal{M} = \{S_1, \ldots, S_k\}$ and let $s \notin \bigcup_{i=1}^{k} S_i$. Then, $\mathcal{M}$ has as many maximal independent sets as $\mathcal{M}' = \{S_1 \cup \{s\}, \ldots, S_k \cup \{s\}, \{s\}\}$; in fact, M is a maximal independent set of $\mathcal{M}$ iff it is also a maximal independent set of $\mathcal{M}'$. We construct a TBox $\mathcal{T}$ from $\mathcal{M}'$ as in the proof of Theorem 6.10. Notice that such a reduction is not parsimonious; for every maximal independent set of $\mathcal{M}$ there can be several MaNAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$. Let $\mathcal{T}'$ be a MaNA, and define $M_{\mathcal{T}'} = \{p \mid A \sqsubseteq P \in \mathcal{T}'\}$. If $\mathcal{S}$, defined by Equation (6.1), is a subset of $\mathcal{T}'$, then $M_{\mathcal{T}'}$ is a maximal independent set for $\mathcal{M}'$; otherwise, there is a set $S \in \mathcal{M}'$ such that $S \subseteq M_{\mathcal{T}'}$. In particular, the latter implies that $A \sqsubseteq P_s \in \mathcal{T}'$, where P_s is the concept name representing the new element s used for defining $\mathcal{M}'$. Thus, the number of MaNAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ equals the number of maximal independent sets for $\mathcal{M}$ plus the number of MaNAs that contain the axiom $A \sqsubseteq P_s$. Consider now the TBox $\mathcal{T}_s = \{A \sqsubseteq P_s \sqcap P \mid p \in \bigcup_{i=1}^{k} S_i\} \cup \{\prod_{j=1}^{\ell_i} P_{ij} \sqsubseteq B \mid 1 \leq i \leq k\}$. All the MaNAs for this TBox are MaNAs for $\mathcal{T}$, and contain $A \sqsubseteq P_s$. Thus, if m_1 is the number of MaNAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ and m_2 the number of MaNAs for $\mathcal{T}_s$ w.r.t. $A \sqsubseteq B$, then the number of maximal independent sets for $\mathcal{M}$ equals $m_1 - m_2$. Since both TBoxes can be computed in polynomial time on the size of $\mathcal{M}$, the problem of counting the number of MaNAs for an $\mathcal{HL}$-TBox w.r.t. a subsumption is #P-hard.

For the second part, given a TBox $\mathcal{T}$, there are as many MaNAs for $\mathcal{T}$ w.r.t. $A \sqsubseteq B$ as there are for $\mathcal{T} \cup \{t\}$ w.r.t. $A \sqsubseteq B$ not containing the axiom t if $t := A \sqsubseteq B$. □

With this we finish our analysis of the complexity of finding MaNAs.

6.1.3 Pinpointing Complexity

All the complexity results presented so far correspond to finding the set of all MinAs, or some of its properties, regardless of the method used. In this dissertation we have focused on an indirect method towards this goal, by finding first a pinpointing formula. As described in Section 3.1, there is a direct correspondence between the MinAs and the minimal valuations satisfying the pinpointing formula, by a bijection between the axioms in the input and the variables appearing in the formula. As

it turns out, analogous complexity results hold already for the problem of finding minimal valuations satisfying a monotone Boolean formula or, as it is also called in the literature, finding the prime implicants of a monotone Boolean formula.

It is worth noticing that every valuation $\mathcal{V}$ can be seen as a monotone Boolean formula consisting of the conjunction of the variables appearing in $\mathcal{V}$. Likewise, a set of valuations represents the disjunction of all the valuations appearing in it; that is, a formula in disjunctive normal form. It is easy to see that, given a monotone Boolean formula ϕ, the set of all minimal valuations satisfying ϕ is equivalent to the original formula ϕ. Since the disjunctive normal form of a formula may be exponential in the number of variables appearing in the formula, it follows that there can be exponentially many minimal valuations that satisfy a given monotone Boolean formula. Additionally, there is no output polynomial algorithm that computes all these minimal valuations (unless P=NP), as shown in [BPS07a, EG91] (see also Theorem 6.4), and counting how many of them exist is #P-complete [Val79b]. Analogous complexity results hold for the problem of finding maximal valuations not satisfying the formula.

These hardness results for monotone Boolean formulae open the question of how hard it is to compute the pinpointing formula *per se*. It could still be the case that finding a pinpointing formula is a simple task, and the whole hardness of computing MinAs is pushed to the computation of minimal valuations from it. Unfortunately, known results in the area of monotone complexity show us that this is not the case.[25]

In essence, Karchmer and Wigderson [KW88, KW90] showed that there exist c-properties decidable in polynomial time whose pinpointing formula is superpolynomial in length (see also Section 5 of [BS90]). The problem they use for showing this result is graph reachability. Consider a set of vertices $V = \{v_1, \ldots, v_n\}$, and let the sets $\mathfrak{I} = \mathfrak{T} = \{(v, w) \mid v, w \in V\}$; that is, the inputs and axioms are pairs of vertices. We see each axiom (v, w) as an edge between v and w. Thus, a set of axioms is a graph. The c-property we are interested in deciding is whether, given an axiomatised input $((v, w), \mathcal{T})$, w is reachable from v in the graph $\mathcal{T}$. Notice that this is indeed a c-property that can be decided in polynomial time. The pinpointing formula for this property and the axiomatised input $((v_1, v_n), \mathfrak{T})$ is not representable in polynomial length [KW88, KW90].

This c-property is in fact a special case of subsumption of $\mathcal{HL}$ concept names, where all the axioms are of the form $C \sqsubseteq D$, with C, D concept names. From this it follows that there exist axiomatised inputs whose pinpointing formula w.r.t. subsumption has superpolynomial length.

Theorem 6.18. *Let* $\mathsf{N_C}$ *be a set of concept names,* $\mathcal{T} = \{C \sqsubseteq D \mid C, D \in \mathsf{N_C}\}$, *and* $A, B \in \mathsf{N_C}$. *The pinpointing formula for* $((A, B), \mathcal{T})$ *w.r.t. subsumption cannot be represented in polynomial length in the size of* $\mathcal{T}$.

With this we conclude our study of the general complexity of pinpointing, and turn now our attention to proving our claim from Chapter 3 with respect to undecidability of termination of the pinpointing extension of general tableaux.

[25] Monotone complexity measures the length of a monotone Boolean formula computing a given function.

6.2 Undecidability of Tableaux Termination

We have now shown several results of the hardness of pinpointing-related problems, independent of the method used for solving them. For the rest of this chapter, we focus our attention once more on the tableau-based method. First of all, notice that we have always assumed that the original tableau terminates on every input (see Definition 3.18) and have not dealt with any means to ensure this fact. Even more, we have shown in Example 3.32 that even if we can prove termination of a general tableau, this will not ensure that its pinpointing extension will also run in finite time. To deal with this problem, we introduced the concept of ordered tableaux in Chapter 4 and showed that they, and their pinpointing extensions, are always terminating. It is nonetheless very easy to see that this class does not fully characterize the class of all tableaux having a terminating pinpointing extension. Unfortunately, as we will see now, it is unable to find a method that decides whether a given tableau has a terminating pinpointing extension.

This section has the following structure. First, we will show that there is a tableau S for which, given an axiomatised input Γ, it is undecidable whether S terminates on Γ by a reduction from the *halting problem* for Turing machines. We then show how to modify the same ideas to show that there is a tableau S for which it is undecidable whether its pinpointing extension terminates on a given axiomatised input Γ. In the end we show how this result relates to our problem of termination in general.

Definition 6.19 (Turing machine). *A* Turing machine *(TM) is a quadruple of the form* $M = (Q, \Sigma, \delta, q_0)$ *where* Q *is a finite set of* states , Σ *is a finite set of* tape symbols *containing the* blank symbol $\sqcup$, $q_0 \in Q$ *is the* initial state *and* δ *is the* transition function $\delta : Q \times \Sigma \to (Q \cup \{\mathsf{yes}, \mathsf{no}\}) \times \Sigma \times \{\leftarrow, \rightarrow\}$. ■

Given a TM M and an input ω, the *halting problem* consists on deciding whether M halts on ω; that is, whether a sequence of computations following the transition relation over the input ω will reach a state where no further step is possible. This problem is well known to be undecidable [Tur36, Dav04]; in other words, there is no algorithm that can decide whether M halts on ω for all possible TMs M and inputs ω. In fact, the following stronger result can be shown: there is a TM M for which the problem of deciding, given an input ω, whether M halts on ω is undecidable. The difference between these two problems is that in the first one the TM is also a part of the input for the decision problem, while in the second one such TM is fixed. Obviously, if there is no algorithm for deciding halting of inputs for a fixed TM, then there is also none that can decide the problem for all possible TMs. We require the stronger result since it is possible to think of general tableaux that are not describable in a finite way, and hence cannot be considered part of the input of a decision procedure.

6.2.1 Termination of Tableaux

We will construct, given a TM M, a tableau S_M whose inputs will be analogous to those of M and such that S_M terminates on an input ω if and only if M halts on

ω. Intuitively, the S_M-states will represent configurations on the tape of M during the execution of the TM and thus a rule application on S_M will simulate the changes performed on the tape by an execution step on M.

We will first use a predicate symbol to represent each symbol in Γ; that is, for every $g \in \Gamma$, include in the signature the unary predicate symbol T_g. To show that the symbol g appears on the tape in the current configuration, we simply use the assertion $T_g(a)$ for some constant a. Since the order in which the symbols appear in the tape is relevant for the execution of the machine, we have to represent such an order accordingly in our tableau states. As S_M-states are merely sets of assertions, we will use predicate symbols F_z for $z \in \mathbb{Z}$ in our signature. Intuitively, an assertion of the form $F_z(a)$ states that constant a is alloted in the tape position z. Such a constant a works as the fusion point between the symbols in the tape and the position they occupy. Thus, we will use distinct constants at different positions.

Once we know the symbols appearing on the tape and their position, we still need to represent the position of the head and the internal state of the machine. We do this in the same way as when describing the tape. For each state $q \in Q$, we add the unary predicate symbol H_q to the signature of our tableau. The assertion $H_q(a)$ represents then that the machine has the internal state q. To know the position to which the head is pointing, we need to look into an assertion of the form $F_z(a)$; this way we know that the head is currently reading the symbol on the z-th cell of the tape.

Example 6.20. *In the initial configuration of the execution of a TM, the head is located in position* 1 *and the internal state is set to* q_0*. Suppose that the input is given by the chain* $s \cdot t$*. This configuration can be represented by the set of assertions*

$$\{F_1(a), F_2(b), T_s(a), T_t(b), H_{q_0}(a)\}.$$

■

Since we want the evolution of the states of our tableau to simulate the transitions performed by the original TM, we need to define the tableau rules accordingly. Suppose, for example, that we have $\delta(q_0, s) = (q_1, s', \rightarrow)$. Given the configuration of Example 6.20, this machine would change the tape to contain the chain $s' \cdot t$, with the head pointing to the second cell and the internal state being q_1. Thus, we would like our rule to change the set of assertions accordingly, leading to the set

$$\{F_1(c), F_2(b), T_{s'}(c), T_t(b), H_{q_1}(b)\}.$$

It is very easy to add the required assertions with a rule application. Unfortunately, tableau rule applications only extend the sets of assertions, and never remove elements from it. Since we have used distinct constants for representing distinct positions (i.e., cells) of the tape, we can add an assertion specifying that a given constant should not be considered anymore as part of the description of a configuration of the TM. We achieve this with the aid of the additional unary predicate $\perp$ in the signature of S_M. We can now describe the configuration after one execution step in M with the set of assertions

$$\{F_1(a), T_s(a), H_{q_0}(a), \perp(a)\} \cup \{F_1(c), F_2(b), T_{s'}(c), T_t(b), H_{q_1}(b)\}.$$

The second set in this expression contains all the elements representing the actual configuration of the machine, the first set showing all the elements that are related to the constant a, which is discarded by the assertion $\bot(a)$. This first set can be thought of as a *trash tail* representing the states and symbols that have been overridden during the execution of M.

Suppose now that $\delta(q_1, t) = (q_1, t, \rightarrow)$. When the machine executes this transition, the head ends up pointing at a cell that is empty and not represented in the S_M-state. Since we cannot know beforehand how many tape cells will be used during the execution of M, we cannot represent all of them in the tableau state either. What we need is a way of expanding the space *on demand*. In this example, we need to specify that the third tape cell will also be used. Thus, we need to add an assertion of the form $F_3(d)$. Furthermore, we know that the tape is written with the blank symbol $\sqcup$ at that cell, so we also include the assertion $T_\sqcup(d)$.

This approach, nonetheless, requires us to know that there is no assertion of the form $F_3(x)$ before the rule can be applied; otherwise, this rule could also be applied to "add space" that is already in use. For example, one such rule application could add the new assertions $F_2(e), T_\sqcup(e)$, which is an undesired behaviour. Our definition of rule application does not allow to look for the (non-)existence of an assertion of some shape; nonetheless, we will be able to do such a check indirectly by using non-deterministic rules. One of the non-deterministic options will try only to add an assertion $F_n(y)$, while the other will add both necessary elements, namely $F_n(y), T_\sqcup(y)$. The way rule application was defined ensures that this rule is only applicable if there is no assertion specifying the use of space in cell n already. We are now ready to construct our tableau S_M that simulates the TM M.

Definition 6.21 (Simulating tableau). *Let $M = (Q, \Gamma, \delta, q_0)$ be a TM and let the set of inputs $\mathfrak{I} \subseteq \Gamma^*$ and set of axioms $\mathfrak{T} = \emptyset$. The* tableau simulating M *is the tableau for $\mathcal{I}$ and $\mathcal{T}$ given by $S_M = (\Sigma, \cdot^S, \mathcal{R}, \mathcal{C})$ where*

- $\Sigma = \{F_z \mid z \in \mathbb{Z}\} \cup \{T_g \mid g \in \Gamma\} \cup \{H_q \mid q \in Q\} \cup \{\bot\}$, *all of arity* 1*;*
- *for every $w = g_1 \cdots g_k \in \mathcal{I}$, we have*

$$w^S = \{T_{g_i}(a_i), F_i(a_i) \mid 1 \leq i \leq k\} \cup \{H_{q_0}(a_1)\};$$

- *for every pair $(q, g) \in Q \times \Gamma$, if $\delta(q, g) = (q', g', \rightarrow)$, then the rules*

$$\begin{aligned} (\{F_k(x), T_g(x), H_q(x), F_{k+1}(y), S_{g''}(y)\}, \emptyset) &\rightarrow \{\{F_k(z), T_{g'}(z), H_{q'}(y), \bot(x)\}\} \\ (\{F_k(x), T_g(x), H_q(x)\}, \emptyset) &\rightarrow \{\{F_{k+1}(z)\}, \{F_{k+1}(z), T_\sqcup(z)\}\} \end{aligned}$$

are in $\mathcal{R}$, and if $\delta(q, g) = (q', g', \leftarrow)$, then the rules

$$\begin{aligned} (\{F_k(x), T_g(x), H_q(x), F_{k-1}(y), S_{g''}(y)\}, \emptyset) &\rightarrow \{\{F_k(z), T_{g'}(z), H_{q'}(y), \bot(x)\}\} \\ (\{F_k(x), T_g(x), H_q(x)\}, \emptyset) &\rightarrow \{\{F_{k-1}(z)\}, \{F_{k-1}(z), T_\sqcup(z)\}\} \end{aligned}$$

are in $\mathcal{R}$, for all $k \in \mathbb{Z}$; and

- $\mathcal{C} = \emptyset$.

■

Theorem 6.22. *Let M be a TM, S_M its simulating tableau and w an input for M. Then, M halts on w if and only if S_M terminates on the axiomatised input $(w, \emptyset)$.*

Proof. At every S_M-state, at most one rule is applicable, described by the only assertion of the form $H_q(a)$ such that there is no assertion $\bot(a)$ in the same state. There are two kinds of applicable rules: those that correspond to a transition of the original TM, which are deterministic, and the non-deterministic ones used to expand the description of the tape. If one of the former kind is applied, then the assertion $\bot(a)$ is added, as well as a new assertion $H_{q'}(b)$, pointing to the new tape cell where the rule applies. The new S_M-state obtained this way represents the configuration on the tape after the TM transition is applied. If the non-deterministic rule is applied, then we obtain two new S_M-states. The first one, in which only an assertion $F_k(c)$ is added, is already saturated, and hence is irrelevant in the search of termination. The second one changes in no way the description of the tape, but allows the rule of the first kind to be triggered. Thus, every S_M state represents a reachable configuration of the TM M over input w. Likewise, for every reachable configuration, there is a S_M-state that represents it. □

Corollary 6.23. *There is a tableau S for which it is undecidable whether it terminates over a given axiomatised input.*

Notice that the simulating tableau does not have any axioms in its inputs. This means that the pinpointing extension of a simulating tableau corresponds to the same original tableau. Hence, we have also shown that there is a tableau for which it is undecidable whether its pinpointing extension terminates on a given input w. But it is still possible to ask about the pinpointing extension of *terminating* tableaux as we do in the following subsection.

6.2.2 Pinpointing Extensions of Terminating Tableaux

We will show now that there exists also a terminating tableau for which it is undecidable whether its pinpointing extension terminates on a given axiomatised input. For this, we want now to construct a terminating tableau whose *pinpointing extension* simulates the TM. One thing to notice first is that none of the rules of the tableau simulating a TM described before can be applied if there is no assertion of the form $H_q(x)$ representing the internal state of the machine. Thus, if we could leave out all these assertions, the TM behaviour will not be simulated by the tableau. The idea is then to create a tableau that starts with a state describing the whole input, but not the initial internal state of the machine, which we know that must be q_0. This tableau should then never add the assertion $H_{q_0}(a_1)$ to the states, ensuring that the simulating rules are not triggered. Additional rules in this tableau should ensure that, when executed in the normal way, it always terminates, but when executing its pinpointing extension, using some axioms, the assertion $H_{q_0}(a_1)$ is added and then the

TM is simulated. This way, we will have a terminating tableau S_M whose pinpointing extension terminates on an input $(w, \mathcal{T})$ if and only if the TM M halts on input w.

To do this, we first allow the set of axioms to be $\mathfrak{T} = \{\mathrm{ax}_1, \mathrm{ax}_2\}$, with all its subsets being admissible. Then, we modify the tableau S_M of Definition 6.21 to construct S'_M in the following way. Add to the signature Σ the unary predicate names P, P', P_1, P_2; and add to $\mathcal{R}$ the rules

$$(\{P(x)\}, \{\mathrm{ax}_1\}) \rightarrow \{\{P'(x), P_1(x)\}\} \tag{6.2}$$

$$(\{P(x)\}, \{\mathrm{ax}_2\}) \rightarrow \{\{P'(x), P_2(x)\}\} \tag{6.3}$$

$$(\{P'(x)\}, \emptyset) \rightarrow \{\{H_{q_0}(x)\}, \{P_1(x)\}, \{P_2(x)\}\}. \tag{6.4}$$

Furthermore, we modify the definition of $\cdot^S$ to replace $H_{q_0}(a_1)$ by $P(a_1)$ in w^S.

The new tableau formed this way is clearly terminating. At the initial state, none of the rules for simulating the TM can be triggered, since $H_{q_0}(a_1)$ is not present. The only way to add this assertion is to apply Rule (6.4), which in turn can only be applied once an assertion of the form $P'(x)$ is present; that is, after applying either Rule (6.2) or Rule (6.3). But once any of these rules is applied, the applicability conditions of non-deterministic rules disallow the possibility of Rule (6.4) to be applied. Hence, after at most two rule applications (depending on the set of axioms given in the axiomatised input), this tableau reaches a saturated state. We thus conclude that S'_M terminates on every axiomatised input.

On the other hand, if the pinpointing extension of S'_M is applied with an input containing both axioms ax_1 and ax_2, then after the application of both Rules (6.2) and (6.3), Rule (6.4) becomes pinpointing applicable. This is the case because the label of the assertion $P'(a_1)$ at this point is $\mathrm{ax}_1 \vee \mathrm{ax}_2$, which does not imply the label of any of the assertions $P_1(a_1)$ or $P_2(a_1)$, given by ax_1 and ax_2, respectively. After applying this rule, we obtain three S'_M states. Two of them, those corresponding to the last two sets in the rule, are already saturated, but not the third one. The third S'_M-state now contains the assertion $H_{q_0}(a_1)$, the only missing piece to start the simulation of the TM over the same input given. Thus the pinpointing extension of S'_M terminates on an input $(w, \mathfrak{T})$ if and only if M halts on w, which gives us our desired undecidability result.

Corollary 6.24. *There is a terminating tableau S for which it is undecidable, given an axiomatised input Γ, whether the pinpointing extension of S terminates on Γ.*

Notice that this is not exactly the result we are looking for. We would like to be able to classify all the tableaux whose pinpointing extension terminates on *all* inputs. It could be the case, for example, that every terminating tableau for which the undecidability result of Corollary 6.24 holds has also an axiomatised input for which non-termination can be decided. We could then still be able to find all the tableaux we are interested in. This, unfortunately, is not the case, given the fact that we can choose the set of inputs over which the tableau can be applied. Define then, for a given tableau S over $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$ and an axiomatised input $\Gamma = (\mathcal{I}, \mathcal{T})$, the *restricted tableau* S_Γ over $\mathfrak{I}' = \{\mathcal{I}\}$ and $\mathscr{P}_{admis}(\mathfrak{T}') = \{\mathcal{S} \in \mathscr{P}_{admis}(\mathfrak{T}) \mid \mathcal{S} \subseteq \mathcal{T}\}$. Then, S_Γ terminates on all axiomatised inputs if and only if S terminates on input Γ.

We have thus shown that it is impossible to fully characterize the set of all tableaux that have a terminating pinpointing extension. This obviously does not mean that we cannot find other subclasses, or even further extend the class of ordered tableaux introduced in Chapter 4, but that there is no way of describing all the elements of the class.

This finishes our study of the complexity of pinpointing, and with it, the whole body of this dissertation. This chapter has shown us that the problem of pinpointing, with all the tasks that surround it, is in general a hard one. For the constructive decision procedures, characterised in this work by general tableaux, the pinpointing extension follows a very intuitive construction, as witnessed by the different times when these ideas have appeared, presumably in an independent way. But our undecidability results show that, although the pinpointing extension is simple, special attention has to be paid, lest we end up with an algorithm that runs indefinitely. But the problems are not inherent to the tableau-based approach. As our general complexity results show, it is simply not possible to design any algorithm that will behave nicely complexity-wise for solving the problems of pinpointing (unless P=NP).

We based our complexity results on the justification and diagnoses problems w.r.t. $\mathcal{HL}$ knowledge bases. This was motivated by the polynomial complexity of its decision problem. Unfortunately, this leaves some problems open. For one, the reductions presented rely on having a general $\mathcal{HL}$-TBox. In description logics, it is sometimes the case that the use of an acyclic TBox allows for a lower complexity bound. From our present study, it is still unclear whether this is the case for pinpointing in the logic $\mathcal{HL}$ or not. Another interesting problem left open during this chapter is the exact complexity of enumerating all MinAs (or MaNAs) if we do not allow for the more general languages used in our proofs.

In the next and closing chapter we give our conclusions as well as some of the open questions that were left by the present dissertation, including those described above. These open problems could be used as starting points for further research in the area of pinpointing.

Chapter 7

Conclusions and Future Work

In this closing chapter we present first a chronical summary of the dissertation, including some concluding remarks and brief insights about the process that led to some of the results readily presented. This summary is followed by some ideas of possible future work that can be built over the results included in this and other related works. Some problems left open by this dissertation are also included.

7.1 A Chronical Summary

In different areas, the need to understand the influence of portions of a theory over the consequences it produces has arisen as a natural problem with distinct applications. This understanding is usually achieved through the computation of one or several MinAs. There are essentially two methods to find these sets, once that a decision procedure exists: one can either call the decision procedure *as-is* using different portions of the theory (the black-box approach), or one can try to modify the original algorithm in such a way that a single execution shines some light on the influence particular axioms have over the result (the glass-box approach).

In this work we had a look at how a glass-box approach works if our decision procedure is either tableau- or automata-based. Although it is possible to think of decision algorithms that do not fall into any of these two categories, these are in reality very rarely found in logic, specially when dealing with monotone properties, which is one of our most basic assumptions.

Very recently, the problem of finding MinAs started to gain relevance in the area of Description Logics, where it got the name of *axiom pinpointing*. The first studies of this problem in DLs produced a custom modification of a tableau-based decision algorithm, which allowed to find one or (a description of) all MinAs for the studied consequence. All these custom modifications had several elements in common, mainly by the tracing mechanism they implemented. Nonetheless, it was not completely obvious how the same ideas would apply to different constructors and their associated tableau rules. Hence, each particular pinpointing extension had to be tested for correctness individually. This motivated our quest for a general notion of tableau-based axiom pinpointing, from which different instances can be taken and known to be

correct without the hassle of solving similar problems once and again.

In order to describe a general approach to tableau-based pinpointing we faced first the task of formalising the notion of a tableau algorithm. Although the main ideas of this class of algorithms seem in general intuitive, there have been very few attempts to formalise them. This corresponds perhaps to the fact that the intuitive notion is so vague as to allow for a perfect formal fitting: any definition would either exclude relevant examples, or be too broad, allowing for techniques that are not generally considered to be tableau-based. Our formalisation is no exception of this. In particular, instances of what we call ground tableau (e.g. the subsumption method for $\mathcal{EL}$, or the congruence closure algorithm) do not seem to be considered as tableau-based by the community. In the other direction, even trying to be as general as possible, our current approach cannot deal with complex blocking techniques, like the ones used for DLs with number restrictions to ensure termination of the process.

With a general notion of tableau, we could then proceed to define their pinpointing extensions in a way that would allow us to compute all the MinAs for a property, represented in a monotone Boolean formula. Our method follows the ideas previously presented in the DL community, but generalises them in a way that allows for distinct kinds of rules and structures that have not previously been considered. For instance, our pinpointing extension can be used alongside with ternary predicates, while DLs deal usually only with unary (concepts) and binary (roles) predicates. There were nonetheless unexpected problems during the development of our framework.

For one, we must speak of the problem of termination of the pinpointing extensions of general tableau. In the original sources motivating our generalisation, termination of the pinpointing extension was disregarded as a trivial consequence of termination of the original tableau algorithm. Intuitively, it indeed looks so, and in a first approach we thought that termination of pinpointing extensions should as trivially follow in the general case. As we saw at the end of Chapter 3, this intuition was incorrect, as multiple applications of rules, caused by the need to understand *all* causes for the insertion of a given assertion, may result in a combination that leads to non-termination. Such a behaviour does not seem to occur in the tableaux for DL.

To recover termination we looked again at successful cases and distinguished, as others have done before, the tree-shape of the created structures as a common cause for termination. That lead to the definition of forest tableaux which, under some additional assumptions, were shown to terminate. Even if they do not satisfy the assumptions required for termination, we showed that equality blocking can be used in this setting to obtain effective algorithms. Tree tableaux obviously constitute a very small subclass of general tableaux, and its definition may seem too complex. In reality, although several conditions are imposed to these tableaux, all of them are syntactical, on the shape of the rules. This might very well exclude several other terminating pinpointing extensions, but syntactical restrictions have the advantage of being easily verified for any given tableau. Other notions of terminating tableaux may possibly be defined, but we showed that it is impossible to fully characterise this class.

While researching in this topic, we slowly became aware of the fact that the same ideas had appeared often in other areas. Particulary surprising is the fact that all glass-box methods found followed the same pattern: the implementation of a tracing

technique over a constructive algorithm. Here the term *constructive* refers to the fact that these algorithms use rules and axioms to construct some kind of model from which the property can be readily decided. The tracing technique consists on adding a label to each piece of this model, which expresses the causes for its addition. This label is modified if more causes become known.

Automata-based decision procedures are not constructive. In their most basic formulation, we construct an automaton based on the input of the problem. The input is rejected if and only if this automaton has a successful run with the root label belonging to the set of initial states. Nonetheless, trying to build a successful run leads, in the best case, to a non-deterministic procedure. This can be improved for automata, by means of an iterative emptiness test, that runs in (deterministic) polynomial time in the size of the automaton. Such a test tells us only whether the language accepted by the automaton is empty or not, but tells us nothing about how this language (or, more correctly, the accepting runs) looks like. Hence, although we can correctly decide a property, it is not simple to trace the reasons of this decision back to the original axioms. This difficulty is further augmented by the fact that the function mapping inputs to automata may actually be arbitrary, holding no regularities with respect to the axioms employed.

Given the prominence of automata-based decision procedures in DLs for showing complexity, and their practical use in some temporal logics, where they have been successfully implemented, it seemed only natural to try to find a way to compute the pinpointing formula from an automata-based decider. The first step was to force a regularity that could allow us to reason about particular axioms. This was done through the definition of axiomatic automata, which states that the addition of new axioms can only restrict the set of successful runs and initial states. The only step left consisted in finding a way to modify the original automaton, or its emptiness test, to compute a monotone Boolean formula, rather than just a yes/no answer. Weighted automata came out as a direct solution: they extend automata theory to the computation of values of a semiring. We showed how to transform an axiomatic automaton into a weighted automaton whose behaviour corresponds to the pinpointing formula. At this time we were surprised not to find any algorithm for computing the behaviour of weighted automata of the kind we were dealing with, and so, developed one of our own by generalising the well-known iterative emptiness test. One thing to notice is that the emptiness test relies heavily on the distributivity of the logical operators over each other. For the general case, such distributivity could not lose its importance, and hence our algorithm could only work on distributive semirings. As every distributive semiring is in fact a lattice, our formulation requiring weights to belong to a distributive lattice is in fact the weakest we could allow in our setting. With this restriction, we were able to prove the correctness of an algorithm that finds the behaviour using time polynomial in the size of the original automaton.

By the time we were finishing our research on the computation of behaviour of automata, we became aware of a different method, developed independently, for solving the same task. However, when we analysed how this method reduces to the case of pinpointing, which was the main concern of our study, we found out that the alternative method is equivalent to the most naïve black-box approach, in which every

possible set of axioms is tested for the property, and then the minimal ones are taken as MinAs. With that in mind, we constructed some examples where our method performs exponentially faster than the other one.

Up to this point, except for the upper-bound obtained by pinpointing automata, there was no clear understanding of the complexity of finding MinAs, or their associated problems. We went on to show that, in general, pinpointing is a hard problem. Although in the logic $\mathcal{HL}$ finding one arbitrary MinA is feasible, as well as finding the last lexicographical one, this positive track disappears once we want to find additional properties that shine some light over the set of all MinAs. Their dual properties are also hard for the set of MaNAs. Furthermore, even the most compact representation of these sets as a monotone Boolean formula may be superpolynomial in length. Notice that this result does not violate the one saying that automata compute the pinpointing formula in polynomial time in the size of the automaton, as we employed a different representation formalism, namely structure sharing, to obtain the feasible time-bound.

7.2 Future Work

One of the main motivations for this work, as has been previously repeated, was the search for a general description of the glass-box strategies used for pinpointing in Description Logics. Our framework is, not surprisingly, general enough to be applied to other settings. One obvious example is the use of the temporal logic LTL as an example for the need of generalised Büchi automata, in Chapter 5. This suggests that there is still a wide range of related problems that can be studied. We present here some of these problems, in most cases accompanied by some thoughts on how can they be approached.

The first and most obvious problem concerns a better understanding of the pinpointing extension of general tableaux, specially regarding their execution time. We know that in general it is impossible even to ensure a finite execution time; but even when the pinpointing extension is known to terminate, there is no appropriate bound on the number of rule applications that are necessary before a saturated state is reached. In the case of ground tableaux, it is easy to see that an exponential blowup in the number of rule applications cannot be avoided in the general case. This follows from the fact that rule applications may modify the label of a single assertion from the least- to the most-general monotone Boolean formula in exponentially many steps. Conversely, it is a very simple exercise to show that such an exponential blowup yields an upper bound on the execution time of the pinpointing extension. We will return to this later on, when we speak about lattices. Once we introduce variables, though, this count becomes much more complicated. Rule applications can still modify the label of a single assertion at most exponentially many times, but additional rule applications may cause the inclusion of new assertions that would never appear during the regular tableau execution. It is not clear how many of these new assertions will be introduced, even for ordered forest tableaux.

Continuing in the complexity line of thought, we have left some unsolved prob-

lems in this work. With respect to the complexity of enumerating all MinAs and/or MaNAs, our hardness results are weaker than desired, as we assumed that a portion of the ontology is composed of axioms that cannot be refuted for the computation of justifications or diagnoses. It is not very clear how to remove this generalisation. In fact, it seems that allowing an irrefutable set of axioms suffices to show hardness: in [Bie08] it was shown that there is no output polynomial algorithm for enumerating all MinAs even if the refutable axioms and the subsumption being justified are all of the form $\top \sqsubseteq A$, where A is a concept name.[26] Most of our MinA-related complexity results rely on a reduction from the minimal hitting set problem. Unfortunately the exact enumeration complexity of the hitting set problem is a long-standing open problem. In Section 6.1.1 we have shown that enumerating all MinAs is at least as hard as enumerating all minimal hitting sets. Our claim is that a reduction in the other direction is not possible, ruling out the equivalence of both problems.

Another problem that was left unsolved is the complexity of pinpointing on *acyclic* TBoxes. All our hardness results for $\mathcal{HL}$ depend on the use of a set of GCIs that does not satisfy the acyclicity assumption. In DL, reasoning under acyclic TBoxes can sometimes lead to a lower complexity class, as attested by the logics $\mathcal{ALC}$ and $\mathcal{SI}$. It could still be the case that feasibility can be regained in $\mathcal{HL}$ in this restricted setting. Likewise, our automata-based approach can be used to prove an exponential upper bound for pinpointing in $\mathcal{SI}$ with acyclic TBoxes, but it is not clear that the bound is tight. For deciding a property, we have shown that a (non-deterministic) top-down emptiness test can sometimes be used to find a tighter upper bound [BHP08]. It is, however, unclear how the same ideas could be applied to pinpointing as the top-down approach yields the information of only one successful run, while pinpointing needs to be able to reason about all of them.

One can also consider finding approximate solutions to some of the problems. Consider for instance the problem of finding the MinA with the least axioms; this is an important problem as small MinAs are usually easier to understand. We have shown that finding the smallest MinA is an NP-hard problem, but it is perhaps possible to construct a procedure that approximates its solution. Such a procedure should compute in polynomial time a MinA whose size is guaranteed to diverge only slightly from the optimal. Alternatively, stochastic methods can be used to find MinAs having a high probability of being optimal. Other problems whose approximation could be of interest include computing the lexicographical first MinA or the total number of MinAs.

For our automata-based approach to pinpointing, we had first to identify contributions of individual axioms to the property under consideration. To this end, we defined mappings that yield, for every axiom t, those initial states and transitions that are allowed by the use of t in the ontology. A more general framework could also allow axioms to influence the acceptance condition of the axiomatic automaton. Such a generalisation was in part left out of this work due to our lack to conceive any scenario that could motivate its application. Another possible generalisation of

[26] In reality, the reduction presented in [Bie08] shows hardness for the DL $\mathcal{EL}$, that is, with the help of existential restrictions. It is nonetheless not hard to adapt the same reduction to the logic $\mathcal{HL}$, thus obtaining a result more akin to those in this dissertation.

the automata-based framework consists in including more general classes of automata. For instance, it seems likely that an algorithm similar to ours can be used to compute the behaviour of weighted parity automata. Apparently, if the automaton is such that the acceptance condition can be tested locally, by the construction and concatenation of finite runs, its behaviour can be computed by an iterative algorithm akin to the one presented in Chapter 5.

Pinpointing, as described in this dissertation, creates a bijection between axioms and a set of propositional variables that will be used to describe the pinpointing formula. As the automata-based approach teaches us, the propositional variables and all the monotone Boolean formulas constructed over them are in fact elements from a free distributive lattice. One can thus think of applying the same ideas using different lattices: we map each axiom to an element of the lattice; this mapping is then used to construct a weighted automaton whose behaviour yields a desired value. Preliminary work on this topic has shown that it may be necessary to restrict the mapping to obtain meaningful results. Of course, such a scenario is not limited to the automata approach, as it is also possible to conceive the development of *weighted tableaux* from the same line of thought. So far as weighted tableaux follow the same ideas of pinpointing extensions, all our results can be reused; for example, one can show that for ground tableaux, the weighted extension will have an overhead execution time proportional to the longest chain of the form $0 < s_1 < \ldots < s_n < 1$. Unfortunately, the negative results and in particular all the problems related to termination, would be still present in this setting.

One possible application of the weighted scenarios just described corresponds to reasoning under vagueness. Indeed, some of the norms used in the definition of fuzzy constructors generate distributive lattices. If instead one was interested in reasoning with probabilities, then more work needs to be done. For some applications, one is interested in knowing whether one can construct a model for a property with probability 1. In this particular case, it would suffice to use the so-called probability semiring, that in fact computes the maximum probability of sequences of independent events. But the probability semiring is not distributive, and hence it is not clear whether the weighted approach can correctly be applied to it. Evenmore, if one wanted to actually compute the probability of a property to follow, one would instead need to reason with measures, which are more complex algebraic structures.

Modern reasoners for DLs, which are tableau-based, rely on heavy optimizations to produce an answer in a timely manner. Our description of the pinpointing extension requires several of these optimizations to be shut off; otherwise, correctness cannot be guaranteed. This is perhaps one of the reasons why in recent time much attention has been paid to black-box pinpointing. A study of new optimization strategies that can also be applied for pinpointing would very likely improve the practicality of the task.

As it can be seen, there is still much work that can be built over the results and ideas of this dissertation. This is hardly surprising, since the problems of finding justifications and diagnoses are relevant in several fields, as attested by the section on related work. This makes the search of general methods, that can be shared between different fields, and possibly using distinct decision procedures, more relevant.

Bibliography

[ACGM04] Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, and Marco Maratea. A SAT-based decision procedure for the boolean combination of difference constraints. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, BC, Canada, 2004. Cited in page(s) 10

[Baa03a] Franz Baader. The instance problem and the most specific concept in the description logic $\mathcal{EL}$ w.r.t. terminological cycles with descriptive semantics. In Andreas Günter, Rudolf Kruse, and Bernard Neumann, editors, *Proceedings of the 26th Annual German Conference on Artificial Intelligence, KI 2003*, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 64–78, Hamburg, Germany, 2003. Springer-Verlag. Cited in page(s) 18

[Baa03b] Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 319–324. Morgan Kaufmann, 2003. Cited in page(s) 18

[Baa03c] Franz Baader. Terminological cycles in a description logic with existential restrictions. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 325–330, Acapulco, Mexico, 2003. Morgan Kaufmann, Los Altos. Cited in page(s) 3

[BBC+05] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani. An incremental and layered procedure for the satisfiability of linear arithmetic logic. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, volume 3440 of *Lecture Notes in Computer Science*, pages 317–333. Springer-Verlag, 2005. Cited in page(s) 10

[BBH96] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996. Cited in page(s) 67

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 364–369, Edinburgh (UK), 2005. Morgan Kaufmann, Los Altos. Cited in page(s) 7, 19

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003. Cited in page(s) 3, 11, 12, 104

[BDS93] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993. Cited in page(s) 23

[BH91] Franz Baader and Philipp Hanschke. A schema for integrating concrete domains into concept languages. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 452–457, Sydney, Australia, 1991. Morgan Kaufmann, Los Altos. Cited in page(s) 19

[BH95] Franz Baader and Bernhard Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning*, 14:149–180, 1995. Cited in page(s) 8, 33, 36

[BHP07] Franz Baader, Jan Hladik, and Rafael Peñaloza. SI! automata can show PSPACE results for description logics. In Remco Loos, Szilárd Zsolt Fazekas, and Carlos Martin-Vide, editors, *Proceedings of the First International Conference on Language and Automata Theory and Applications (LATA'07)*, Tarragona, Spain, 2007. Cited in page(s) 28

[BHP08] Franz Baader, Jan Hladik, and Rafael Peñaloza. Automata can show PSPACE results for description logics. *Information and Computation*, 206(9,10):1045–1056, 2008. Special Issue: First International Conference on Language and Automata Theory and Applications (LATA'07). Cited in page(s) 7, 9, 28, 91, 129

[Bie08] Meghyn Bienvenu. Complexity of abduction in the $\mathcal{EL}$ family of lightweight description logics. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'2008)*, pages 220–230. AAAI Press/The MIT Press, 2008. Cited in page(s) 10, 129

[BL85] Ronald J. Brachman and Hector J. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Altos, 1985. Cited in page(s) 140, 141

[BP07] Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. In Nicola Olivetti, editor, *Proceedings of the 16th International Conference on Analytic Tableaux and Related Methods (TABLEAUX 2007)*, volume 4548 of *Lecture Notes in Artificial Intelligence*, pages 11–27, Aix-en-Provence, France, 2007. Springer-Verlag. Cited in page(s) 7

[BP08] Franz Baader and Rafael Peñaloza. Automata-based axiom pinpointing. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2008)*, volume 4667 of *Lecture Notes in Artificial Intelligence*, pages 226–241, Sydney, Australia, 2008. Springer-Verlag. Cited in page(s) 8

[BP09] Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 2009. Special Issue: Tableaux'07. To appear. Cited in page(s) 7

[BPS07a] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}^+$. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *Proceedings of the 30th German Annual Conference on Artificial Intelligence (KI'07)*, volume 4667 of *Lecture Notes in Artificial Intelligence*, pages 52–67, Osnabrück, Germany, 2007. Springer-Verlag. Cited in page(s) 8, 104, 105, 107, 112, 117

[BPS07b] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}$. In Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris, editors, *Proceedings of the 2007 Description Logic Workshop (DL 2007)*, volume 250 of *CEUR-WS*, Brixen-Bressanone, Italy, 2007. Cited in page(s) 8

[Bra04a] Sebastian Brandt. On subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general TBoxes. In Volker Haarslev and Ralf Möller, editors, *Proceedings of the 2004 Description Logic Workshop (DL 2004)*, volume 104. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-104/, 2004. Cited in page(s) 19

[Bra04b] Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, pages 298–302, 2004. Cited in page(s) 3

[Bru05] Renato Bruni. On exact selection of minimally unsatisfiable subformulae. *Annals of Mathematics and Artificial Intelligence*, 43(1):35–50, 2005. Cited in page(s) 10

[BS90] Ravi B. Boppana and Michael Sipser. The complexity of finite functions. In *Handbook of theoretical computer science (vol. A): algorithms and complexity*, pages 757–804, Cambridge, MA, USA, 1990. The MIT Press. Cited in page(s) 117

[BS01] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001. Cited in page(s) 7, 18, 22

[BS05] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In Manuel V. Hermenegildo and Daniel Cabeza, editors, *Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages (PADL'05)*, volume 3350 of *Lecture Notes in Computer Science*, pages 174–186, Long Beach, CA, USA, 2005. Springer-Verlag. Cited in page(s) 9

[BS08] Franz Baader and Boontawee Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proceedings of the 3rd Knowledge Representation in Medicine (KR-MED'08)*, volume 410 of *CEUR-WS*, 2008. Cited in page(s) 4, 9, 105

[BT01] Franz Baader and Stephan Tobies. The inverse method implements the automata approach for modal satisfiability. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 92–106, Siena, Italy, 2001. Springer-Verlag. Cited in page(s) 9, 26

[CD91] John W. Chinneck and Erik W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168, 1991. Cited in page(s) 10

[CDGL99] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proceedings of the 16th Interntional Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 84–89, 1999. Cited in page(s) 9

[CDGL02] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 2ATAs make DLs easy. In *Proceedings of the 2002 Description Logic Workshop (DL 2002)*, pages 107–118, 2002. Cited in page(s) 9

[Chi97] John W. Chinneck. Finding a useful subset of constraints for analysis in an infeasible linear program. *INFORMS Journal on Computing*, 9(2):164–174, 1997. Cited in page(s) 10, 104, 105

[Dav04] Martin Davis. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions.* Dover Publications, Incorporated, 2004. Cited in page(s) 118

[DDB98] Gennady Davydov, Inna Davydova, and Hans Kleine Büning. An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. *Annals of Mathematics and Artificial Intelligence*, 23(3–4):229–245, 1998. Cited in page(s) 10

[dK86a] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, 1986. Cited in page(s) 10

[dK86b] Johan de Kleer. Extending the ATMS. *Artificial Intelligence*, 28(2):163–196, 1986. Cited in page(s) 10

[dK86c] Johan de Kleer. Problem solving with the ATMS. *Artificial Intelligence*, 28(2):197–224, 1986. Cited in page(s) 10

[DK06] Manfred Droste and Dietrich Kuske. Skew and infinitary formal power series. *Theoretical Computer Science*, 366(3):199–227, 2006. Cited in page(s) 80

[DKR08] Manfred Droste, Werner Kuich, and George Rahonis. Multi-valued MSO logics over words and trees. *Fundamenta Informaticae*, 84(3,4):305–327, 2008. Cited in page(s) 9, 76, 83, 98, 102

[DR06] Manfred Droste and George Rahonis. Weighted automata and weighted logics on infinite words. In Oscar H. Ibarra and Zhe Dang, editors, *Proceedings of the 10th International Conference on Developments of Language Theory (DLT 2006)*, volume 4036 of *Lecture Notes in Computer Science*, pages 49–58, Santa Barbara, CA, USA, 2006. Springer-Verlag. Cited in page(s) 80

[DSV08] Manfred Droste, Jacques Sakarovitch, and Heiko Vogler. Weighted automata with discounting. *Information Processing Letters*, 108:23–28, 2008. Cited in page(s) 80

[EG91] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. Technical Report CD-TR 91/16, Christian Doppler Labor für Expertensysteme, TU-Wien, 1991. Cited in page(s) 107, 117

[EG95a] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, 1995. Cited in page(s) 10, 106

[EG95b] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995. Cited in page(s) 107

[EKM61] Rolf Eberle, David Kaplan, and Richard Montague. Hempel and oppenheim on explanation. *Philosophy of Science*, 28(4):418–428, 1961. Cited in page(s) 2

[EM02] Thomas Eiter and Kazuhisa Makino. On computing all abductive explanations. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2005)*, pages 62–67, Alberta, Canada, 2002. AAAI Press/The MIT Press. Cited in page(s) 10

[FGN90] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Hypothesis classification, abductive diagnosis and therapy. In Georg Gottlob and Wolfgang Nejdl, editors, *Proceedings of the International Workshop on Expert Systems in Engineering, Principles and Applications*, volume 462 of *Lecture Notes in Computer Science*, pages 69–78, Viena, Austria, 1990. Springer-Verlag. Cited in page(s) 106

[GHN+04] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188, Boston, MA, USA, 2004. Springer-Verlag. Cited in page(s) 10

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability — A guide to NP-completeness.* W. H. Freeman and Company, San Francisco (CA, USA), 1979. Cited in page(s) 36, 94, 107

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65, Paris, France, 2001. Springer-Verlag. Cited in page(s) 9

[GPSS80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'80)*, pages 163–173, 1980. Cited in page(s) 16

[GPVW95] Rob Gerth, Doron Peled, Mosche Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall. Cited in page(s) 9

[GR90] John Gleeson and Jennifer Ryan. Identifying minimally infeasible subsystems of inequalities. *INFORMS Journal on Computing*, 2(1):61–63, 1990. Cited in page(s) 10

[Grä98] George Grätzer. *General Lattice Theory.* Birkhäuser, Basel, second edition edition, 1998. Cited in page(s) 79, 99

[HB91] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 335–346, Cambridge, MA, USA, 1991. Morgan Kaufmann, Los Altos. Cited in page(s) 67

[Hem65] Carl G. Hempel. *Aspects of Scientific Explanation (and Other Essays)*. Free Press, New York, 1965. Cited in page(s) 2

[HKS05] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The irresistible SRIQ. In *Proceedings of OWL: Experiences and Directions*, Galway, Ireland, 2005. Cited in page(s) 14

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'2006)*, pages 57–67, Lake District, UK, 2006. AAAI Press/The MIT Press. Cited in page(s) 14

[HM01] Volker Haarslev and Ralf Möller. RACER system description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, 2001. Cited in page(s) 9

[HO48] Carl G. Hempel and Paul Oppenheim. Studies in the logic of explanation. *Philosophy of Science*, 15(2):135–175, 1948. Cited in page(s) 2, 3, 5

[Hol96] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):133–157, 1996. Cited in page(s) 8, 19

[Hol97] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997. Cited in page(s) 9

[Hor98] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In Anthony G. Cohn, Lenhard K. Schubert, and Stuart C. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, Trento, Italy, 1998. Cited in page(s) 9, 23

[HPS08] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in owl. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *Proceedings of the 7th International Semantic Web Conference (ISWC'08)*, volume 5318 of *Lecture Notes in Computer Science*, pages 323–338, Karlsruhe, Germany, 2008. Springer-Verlag. Cited in page(s) 8, 10

[HPS09] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for justifications in owl. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 2009 Description Logic Workshop (DL 2009)*, volume 477 of *CEUR-WS*, 2009. Cited in page(s) 10

[HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003. Cited in page(s) 3

[HS99] Ian Horrocks and Ulrike Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999. Cited in page(s) 24, 65, 67

[HS04] Ian Horrocks and Ulrike Sattler. Decidability of SHIQ with complex role inclusion axioms. *Artificial Intelligence*, 160(1):79–104, 2004. Cited in page(s) 3, 14

[HST00] Ian Horrocks, Ulrike Sattler, and Stefan Tobies. Practical reasoning for very expressive description logics. *Journal of the Interest Group in Pure and Applied Logic*, 8(3):239–264, 2000. Cited in page(s) 3, 67

[JYP88] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988. Cited in page(s) 106, 107, 114

[KL07] Orna Kupferman and Yoad Lustig. Lattice automata. In Byron Cook and Andreas Podelski, editors, *Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'07)*, volume 4349 of *Lecture Notes in Artificial Intelligence*, pages 199–213, Nice, France, 2007. Springer-Verlag. Cited in page(s) 9

[KPHS07] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 267–280, Busan, Korea, 2007. Springer-Verlag. Cited in page(s) 9, 105

[KPSG06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in OWL ontologies. In York Sure and John Domingue, editors, *Proceedings of the 3rd European Semantic Web Conference (ESWC'06)*, volume 4011 of *Lecture Notes in Computer Science*, pages 170–184, Budva, Montenegro, 2006. Springer-Verlag. Cited in page(s) 104

[KSS00] Dimitris J. Kavvadias, Martha Sideri, and Elias C. Stavropoulos. Generating all maximal models of a Boolean expression. *Information Processing Letters*, 74(3–4):157–162, 2000. Cited in page(s) 107

[KW88] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Proceedings of the 20th ACM SIGACT Symposium on Theory of Computing (STOC'88)*, pages 539–550, Chicago, Illinois, USA, 1988. ACM Press and Addison Wesley. Cited in page(s) 117

[KW90] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM Journal on Discrete Mathematics*, 3(2):255–265, 1990. Cited in page(s) 117

[Lei97] Alexander Leitsch. *The Resolution Calculus*. Springer-Verlag, 1997. Cited in page(s) 7

[LMP06] Kevin Lee, Thomas Meyer, and Jeff Z. Pan. Computing maximally satisfiable terminologies for the description logic ALC with GCIs. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *Proceedings of the 2006 Description Logic Workshop (DL 2006)*, Lake District, UK, 2006. Cited in page(s) 8, 9, 34

[LS00] Carsten Lutz and Ulrike Sattler. The complexity of reasoning with boolean modal logic. In *Proceedings of Advances in Modal Logic 2000 (AiML 2000)*, 2000. Cited in page(s) 9

[LS04] Inês Lynce and João P. Marques Silva. On computing minimum unsatisfiable cores. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, BC, Canada, 2004. Cited in page(s) 10

[LS05] Mark H. Liffiton and Karem A. Sakallah. On finding all minimally unsatisfiable subformulas. In Fahiem Bacchus and Toby Walsh, editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *Lecture Notes in Computer Science*, pages 173–186. Springer-Verlag, 2005. Cited in page(s) 9, 36

[LST99] Carsten Lutz, Ulrike Sattler, and Stephan Tobies. A suggestion for an n-ary description logic. In Patrick Lambrix, Alex Borgida, Maurizio Lenzerini, Ralf Möller, and Peter Patel-Schneider, editors, *Proceedings of the 1999 Description Logic Workshop (DL'99)*, pages 81–85, Linkoeping, Sweden, 1999. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-22/. Cited in page(s) 56

[Lut99] Carsten Lutz. Complexity of terminological reasoning revisited. In *Proceedings of the 6th International Conference on Logic for Programming*

and Automated Reasoning (LPAR'99), volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 181–200. Springer-Verlag, 1999. Cited in page(s) 14, 20, 21

[Man08] Eleni G. Mandrali. Weighted tree automata with discounting. Master's thesis, Aristotle University of Thessaloniki, 2008. Cited in page(s) 80

[Mil43] John Stuart Mill. *A System of Logic*. John W. Parker, London, UK, 1843. Cited in page(s) 2

[Min81] Marvin Minsky. A framework for representing knowledge. In John Haugeland, editor, *Mind Design*. The MIT Press, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [BL85]. Cited in page(s) 3

[MLBP06] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*. AAAI Press/The MIT Press, 2006. Cited in page(s) 34

[Neb90] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990. Cited in page(s) 14, 20

[NO07] Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. *Information and Computation*, 205(4):557–580, 2007. Cited in page(s) 7, 37

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006. Cited in page(s) 10

[Peñ08] Rafael Peñaloza. Automata-based pinpointing for DLs. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proceedings of the 2008 Description Logic Workshop (DL 2008)*, volume 353 of *CEUR-WS*, Dresden, Germany, 2008. Cited in page(s) 8

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977. Cited in page(s) 16

[Pop35] Karl Popper. *Logik der Forschung*. Springer, Vienna, Austria, 1935. Cited in page(s) 2

[PSK05] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*, pages 633–640. ACM, 2005. Cited in page(s) 8, 34

[Qui67] M. Ross Quillian. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, 12:410–430, 1967. Republished in [BL85]. Cited in page(s) 3

[Rab70] Michael O. Rabin. Weakly definable relations and special automata. In Y. Bar-Hillel, editor, *Proceedings of Symposium on Mathematical Logic and Foundations of Set Theory*, pages 1–23. North-Holland Publ. Co., Amsterdam, 1970. Cited in page(s) 26

[Rah07] George Rahonis. Weighted muller tree automata and weighted logics. *Journal of Automata, Languages and Combinatorics*, 12(4):455–483, 2007. Cited in page(s) 80

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. Cited in page(s) 8, 9, 105

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965. Cited in page(s) 7

[Rym92] Ron Rymon. Search through systematic set enumeration. In Bernhard Nebel, Charles Rich, and William R. Swartout, editors, *Proceedings of the 3rd International Conference on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 539–550, Cambridge, MA, USA, 1992. Cited in page(s) 105

[Sal89] Wesley C. Salmon. *Four Decades of Scientific Explanation.* University of Minnesota Press, 1989. Cited in page(s) 2

[SC85] A. Prasad Sistla and E. C. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32(3):733–749, 1985. Cited in page(s) 102, 103

[SC03] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 355–362, Acapulco, Mexico, 2003. Morgan Kaufmann, Los Altos. Cited in page(s) 8, 33, 34, 36

[Sch91] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991. Cited in page(s) 104

[Sch94] Klaus Schild. Terminological cycles and the propositional μ-calculus. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the 4th International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 509–520, Bonn (Germany), 1994. Morgan Kaufmann, Los Altos. Cited in page(s) 95, 102

[Sch96] Gerhard Schurz. Scientific explanation: A critical survey. *Foundations of Science*, 3:429–465, 1996. Cited in page(s) 2

[Sch05] Stefan Schlobach. Diagnosing terminologies. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, pages 670–675. AAAI Press/The MIT Press, 2005. Cited in page(s) 8, 111

[Sei94] Helmut Seidl. Finite tree automata with cost functions. *Theoretical Computer Science*, 126(1):113–142, 1994. Cited in page(s) 79

[SHCH07] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39(3):317–349, 2007. Cited in page(s) 9, 105, 111

[SP04] Evren Sirin and Bijan Parsia. Pellet: An OWL DL reasoner. In *Proceedings of the 2004 Description Logic Workshop (DL 2004)*, pages 212–213, 2004. Cited in page(s) 9

[Spa05] Kent A. Spackman. Rates of change in a large clinical terminology: Three years of experience with SNOMED clinical terms. In *Proceedings of the 2005 AMIA Annual Symposium (AMIA 2005)*, pages 714–718. Hanley&Belfus, 2005. Cited in page(s) 4

[SPSW01] Michael Q. Stearns, Colin Price, Kent A. Spackman, and Amy Y. Wang. SNOMED clinical terms: Overview of the development process and project status. In *Proceedings of the 2001 AMIA Annual Symposium (AMIA 2001)*, pages 662–666. Hanley&Belfus, 2001. Cited in page(s) 4

[SQJH08] Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. A modularization-based approach to finding all justifications for owl dl entailments. In John Domingue and Chutiporn Anutariya, editors, *Proceedings of the 3th Asian Semantic Web Conference (ASWC'08)*, volume 5367 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2008. Cited in page(s) 105

[SS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991. Cited in page(s) 8, 12

[ST07] Renate A. Schmidt and Dmitry Tishkovsky. Using tableau to decide expressive description logics with role negation. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 438–451. Springer, 2007. Cited in page(s) 9

[ST08] Renate A. Schmidt and Dmitry Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2008)*, volume 5195 of *Lecture Notes in Computer Science*, pages 194–209, Sydney, Australia, 2008. Springer. Cited in page(s) 9

[Sun09] Boontawee Suntisrivaraporn. *Polynomial-time Reasoning Support for Design and Maintenance of Large-scale Biomedical Ontologies*. PhD thesis, Technische Universität Dresden, 2009. Cited in page(s) 3, 9, 105, 107, 112

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955. Cited in page(s) 85

[TMJ96] Mehrdad Tamiz, S. J. Mardle, and Dylan F. Jones. Detecting IIS in infeasible linear programmes using techniques from goal programming. *Computers and Operations Research*, 23(2):113–119, 1996. Cited in page(s) 10, 105

[Tur36] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936. Cited in page(s) 118

[Val79a] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. Cited in page(s) 110

[Val79b] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. Cited in page(s) 110, 111, 117

[Var96] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996. Cited in page(s) 11

[VW84] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. In *Proceedings of the 16th ACM SIGACT Symposium on Theory of Computing (STOC'84)*, pages 446–455, 1984. Cited in page(s) 26

[VW86] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986. A preliminary version appeared in *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing (STOC'84)*. Cited in page(s) 26, 84, 96

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th Annual Symposium*

of Foundations of Computer Science (SFCS'83), pages 185–194, Washington, DC, USA, 1983. IEEE Computer Society. Cited in page(s) 7, 29, 30

[ZM03] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the 2003 Conference on Design, Automation and Test in Europe (DATE'03)*, pages 10880–10885, Munich, Germany, 2003. IEEE Computer Society Press. Cited in page(s) 10